Appendix

The Mousetrap For HTML Designers

1. Audience and Scope

This guide is for those familiar with HTML but not with SML or JavaScript. It will allow taking an existing mousetrap and modifying it so it can be applied to a different site.

2. Operation

The first two sections of *The Mousetrap Programmer's Guide* can be read by a non-programmers for a description of the mousetrap mechanism. "The Mousetrap Effect" describes the action of the mousetrap from a user's point of view and "Operation Overview" discusses the sequence of events which occur when the browser's back button is pressed or a link is clicked. Also, check the "Mousetrap Flow Drawing" on page 16 to see how a mousetraps flow works.

3. Mousetraping In A Nutshell

In a nutshell, to apply the mousetrap to a non-mousetrap sales site, do the following:

1. Copy & rename the necessary files (section 7).
2. Make sure the ..._tail.htm file copied from above has the proper directory for the legal section (see end of section 6).
3. Insert the proper file names in the frameset page (section 5).
4. Copy the SML at the very top of the page and JavaScript between the </title> and </head> from parallel pages in a mousetrap (for example, copy the SML and JavaScript from the Sex Museum softsales page to the new softsales page.
5. Make sure all links in parallel pages have the JavaScript (onClick, onMouseOver, onMouseOut) and SML tags (including the feeders on the links to the registration pages).
6. Copy the SML conditional for the button/submit-button on the registration pages from mousetrap registration pages (see section 8.4).
7. Perform the tests in the "Testing" section (page 11). If there are problems, go to the "Troubleshooting" section (page 13).

IMPORTANT NOTE: When copying anything from an existing mousetrap to a new mousetrap, load the existing mousetrap file in your editor to copy data. DO NOT COPY BY VIEWING PAGE SOURCE IN YOUR BROWSER... SML COMMANDS WILL NOT BE COPIED.

The Mousetrap For HTML Designers

4. Copy Files

The first step to brewing up a mousetrap is to copy files from an existing mousetrap to the new mousetrap directory. If we were copying from Sex Museum, the following files would be copied:

1. mmuse_index.sml
2. mmuse_hidden1.sml
3. mmuse_hidden2.sml
4. mmuse_tail.htm

We would then rename them appropriately. For example, for Video Sex Channels, the frameset page might be renamed "mvsc_index.sml".

5. The Frameset Page

Various file names need to be entered in this page to customize the mousetrap to a particular sales site.

This page typically ends in "index.sml". For example, Sex Museum's frameset page has the filename of "mmuse_index.sml".

5.1.1. Warning Page

At line 5 of the frameset page, the following needs to have the file name entered for the warning page:

```
<$if cond:!.goTo val:.goTo="mmuse_warning.sml"><$endif>
```

The above setting is used for the warning page in Sex Museum.

5.1.2. defaultStatusMsg

This constant needs to be set with the message desired on the bottom of the browser's window. The following default message was used for Sex Museum:

```
defaultStatusMsg="Welcome to Sex Museum";
```

5.1.3. Hidden Pages

Two constants must be changed to reflect the file names of the hidden files. For example, for the Sex Museum, the following names are used:

```
hidden1="mmuse_hidden1.sml";
hidden2="mmuse_hidden2.sml";
```

Also note that the hidden1 file needs to be set in the frameset definition itself at line 84.

5.1.4. Netscape 2 Kludge

The defaultWarpTo constant is used because of a problem with Netscape 2. The following line needs to be set to the "warpTo" value in the first sales page (the page you come to after you click the "Enter" off the warning page).

```
defaultWarpTo="mmuse_survey.sml
```

6. Warning Page

This page contains 4 lines of JavaScript that will occur on all pages (except the registration pages).

```
parent.backTo="mmuse_softsales.sml";
parent.warpTo="mmuse_softsales.sml";
window.defaultStatus=parent.defaultStatusMsg;
parent.loadPage();
```

The "parent.backTo" needs to be set to the page you need to go to when the user hits the browser's back button. The "parent.warpTo" needs to be set to the page you need to go to when the user tries to go to a new location via bookmarks, history list, or typing in a new URL. The "window.defaultStatus" line sets the default of the browser's status line to "defaultStagusMsg" value set in the frameset page. The last line runs the "loadPage()" method in the parent which loads a file into the hidden frame.

All links on the warning page need to have the following line in them:

```
OnClick="parent.clicked(this)"
```

In addition, a mouse over message can be associated with the link by adding the following JavaScript to the link:

```
OnMouseOver="return parent.setStatus('I agree')"
OnMouseOut="return parent.setStatus('')"
```

The footer also needs to manually be added to the warning page and *every other page* in the mousetrap. For example, the following is used on Sex Museum:

```
<#INC."d:/http/sexmuseum/mmuse_tail.htm"#>
<table width=610 border=0>
  <tr>
    <td>
     <ul>
       <pre><#MAC.NBHOST#></pre></font>
     </ul>
    </td>
  </tr>
</table>
</HTML>
<$exit>
```

The line with "d:/http/sexmuseum/mmuse_tail.htm" would change with another site. For example, at XXX Sex Photos, the line would change to "d:/http/xxxsexphots/mxsp_tail.htm".

7. First Sales Page

This page is loaded after "Enter" on the warning page is clicked. The top of this page needs the following SML line:

<#MAC.visit#>

This page has the same JavaScript as the warning page except that the following line is added:

parent.enterClicked=true;

This is a variable to let other pages know that the user has clicked the "Enter" key.

As discussed in the warning page, links to the "age verification" or registration pages need to have the following JavaScript:

OnClick="parent.clicked(this)"

The SML feeders need to be correct on these links as well to ensure proper statistics. The following is a correct feeder example from Sex Museum:

feeder=MMUSE_sales_1_Live-<#MAC.NBHOST#>&<#SID=.#>&bid=<#.bid#>

The link to the members and webmaster areas need to have the following in order to remove the frameset when entering these areas:

target="_top" OnClick="parent.clicked(this)"

Again, all links can make use of the mouse over events for links:

OnMouseOver="return parent.setStatus('Members Enter Here')"
OnMouseOut="return parent.setStatus('')"

8. Registration Page

8.1. SML

The SML for this page should be copied from an existing mouse trap registration page:

```
<$assign val:SID.mm="apps.xpics.com">
<$if cond:.free!=CHECKED cond:.mall!=CHECKED val:.m90=CHECKED><$endif>
<$if cond:.bid val:.acb=.bid><$endif>
```

The second line above sets the default membership to the 90 Day membership.

8.2. JavaScript

The JavaScript for this page is different than previous pages:

```
<script language="JavaScript">
<!-- Script start
   var isSent = 0;
   parent.newWin=true;

  function load()
  {
   window.defaultStatus=parent.defaultStatusMsg;
  }

  function SubmitData( form )
  {
   parent.newWin=false;
   if( !isSent )
   {
    isSent = 1;
    form.submit( );
   }
  }
// Script end -->
</script>
```

8.3. Membership Codes

The following three lines are specific to each site:

```
<INPUT TYPE="radio" NAME="PARM13" VALUE="SEXM-FR" <#.free#>>
<INPUT TYPE="radio" NAME="PARM13" VALUE="SEXM-90" <#.m90#>>
```

The "SEXM-FR" and "SEXM-90" is specific to Sex Museum. For example, the lines in all XXX Sex Photos registration pages are the following:

```
<INPUT TYPE="radio" NAME="PARM13" VALUE="SEXRXX-FR" <#.free#>>
<INPUT TYPE="radio" NAME="PARM13" VALUE="SEXRXX-90" <#.m90#>>
```

8.4. Submit Button

The submit button needs to look something like the following:

```
<$if cond:SID.js>
<CENTER><input type="button" value="Submit Information" onClick="SubmitData( this.form )"></CENTER>
<$else>
<CENTER><input type="submit" value="Submit Information"></CENTER>
<$endif>
```

The SML conditional <$if cond:SID.js> needs to be used on a mousetrap registration pages. The SML conditional <$if cond:MAC.user_agent LIKE "Mozilla/[2-9]"> is **only** used on non-mousetrap registration pages.

9. The Second Sales (Survey) Page

This page has a form on it and has JavaScript which should be copied from another survey page. Currently the information entered in the survey is being thrown away. In development, we have a survey page which will store data in SQL Server. We will describe these two variations and their setup:

9.1. Data Not Stored

For the survey form that throws away user entered data, the following lines are used near the HTML "form" command:

```
1. <form name="qForm" method=post action="mmuse_allsites.sml">

2. <input type=hidden name="feeder" value="MMUSE_survey-<#MAC.NBHOST#>">
3. <input type=hidden name="referer" value="<#SID.referer#>">
4. <input type=hidden name="bid" value="<#.bid#>">
```

The first line above needs to have the appropriate "allsites" registration page. For example, for XXX Sex Photos, the line should have **action="mxsp_allsites.sml"**.

The second line should have the appropriate feeder. For XXX Sex Photos, it would be **value="MXSP_survey-<#MAC.NBHOST#>"**. The remaining lines would not be altered.

9.2. Data Stored

For the survey form that stores user entered data, the following lines are used near the HTML "form" command:

```
1. <form name="qForm" method=get action="http://apps.xpics.com/cgi-bin/survey.pl">

2. <input type=hidden name="feeder" value="MMUSE_survey-<#MAC.NBHOST#>">
3. <input type=hidden name="referer" value="<#SID.referer#>">
4. <input type=hidden name="bid" value="<#.bid#>">

5. <input type=hidden name="table_name" value="survey_response">
6. <input type=hidden name="redirect" value="http://www.sexmuseum.com/mmuse_allsites1.sml">
```

The second line should have the appropriate feeder. For XXX Sex Photos, it would be **value="MXSP_survey-<#MAC.NBHOST#>"**.

The sixth line above needs to have the full URL of the appropriate "allsites" registration page. For example, for XXX Sex Photos, the line should have **value="http://www.xxxsexphotos.com/mxsp_allsites.sml"**.

The other lines don't need to be altered.

10. The Allsites Page

After a user has filled out the survey page and submitted it, they will be directed to the allsites registration page. Using SML conditionals, this page is responsive, depending on the answers to the questions in the survey page.

Nothing in this page needs to be customized for a particular site.

11. SoftSales Page

If the user hits the browser's back button from the warning page, they will get presented with a "soft" sales page. That is the images in this page are softer than those in the sales page. The JavaScript in this page shows a variation over the regular sales page.

If the user presses the back button at the softsales page, they would normally go to a banners page specific to the current site (see next section). However if the user has come to our Warning without a banner ID (a small percentage) they may be under age so we don't want them to go to the banners page. For this reason, the JavaScript on the softsales page is different than the regular sales page.

```
if (parent.acb == "")
{
 parent.backTo= "http://198.168.54.139/thrill-cgi/ad/xpics/ricochet.cgi?xpicsr";
 parent.warpTo="http://198.168.54.139/thrill-cgi/ad/xpics/ricochet.cgi?xpicsr";
 parent.backToTop=true; //blow away frame
 }
 else
 {
   parent.backTo="mmuse_banners.sml";
   parent.warpTo="mmuse_banners.sml";
 }
 window.defaultStatus=parent.defaultStatusMsg;
 parent.loadPage();
}
```

The JavaScript above sends a user without a banner ID to CyberThrill. If they do have a banner ID, they go to the banners page (mmuse_banners.sml). These lines could change depending on who we are directing potentially underage users to.

WARNING: MAKE SURE THE FIRST LINE ON THE REGULAR SALES PAGE:

<#MAC.visit#>

IS NOT ON THE SOFTSALES PAGE!!!

12. Banners Page

12.1. Links

All links in the banners page **need** to have the following:

```
Target="_top" OnClick="parent.clicked(this)"
```

For example, a link to the ass website could look like the following:

```
<a href="http://www.xpics.com/ass/ass.html?acb=acb100001-1100" Target="_top"
OnClick="parent.clicked(this)">
```

12.2. JavaScript

This is the last stop in the mousetrap before the mousetrap is exited. We have two variations in the JavaScript for the banners page:

1. The user always gets directed to CyberThrill (or another PG rated website) with a back button or warp.
2. If the user has hit the enter button and is backing up from the survey page, they can directed to the sales pages of another Xpics site with a back button or warp. If the user has backed off from the warning page (and gone to the softsales page) they will get directed to CyberThrill (or another PG rated website) with a back button or warp.

12.2.1. Always Directed to PG Website

The JavaScript below will always direct the user to the CyberThrill site with the back button or warp:

```
<SCRIPT LANGUAGE="JavaScript">
  parent.backTo="http://198.168.54.139/thrill-cgi/ad/xpics/ricochet.cgi?xpicsr";
  parent.warpTo="http://198.168.54.139/thrill-cgi/ad/xpics/ricochet.cgi?xpicsr";
  parent.backToTop=true; //blow away frame
  window.defaultStatus=parent.defaultStatusMsg;
  parent.loadPage();
</SCRIPT>
```

12.2.2. Not Always Directed to PG Website

The JavaScript below will direct the user to the CyberThrill site if they have not clicked the enter button on the warning page (ie, haven't gone to the first sales page). If they have clicked the enter button, they will be directed to the "non-mousetrap" sales page of the Video Sex Channels" site (see section 15.2 for loading a mousetrap at a sales page instead of the warning page).

```
<SCRIPT LANGUAGE="JavaScript">
 if (!parent.enterClicked)
 {
    parent.backTo="http://198.168.54.139/thrill-cgi/ad/xpics/ricochet.cgi?xpicsr";
    parent.warpTo="http://198.168.54.139/thrill-cgi/ad/xpics/ricochet.cgi?xpicsr";
    parent.backToTop=true; //blow away frame
 }
 else
 {
    parent.backTo="http://www.videosexchannels.com/vsc_sales.sml?bid=<#.bid#>&refer=
    MMUSE_banners.sml-<#MAC.NBHOST#>";
    parent.warpTo="http://www.videosexchannels.com/vsc_sales.sml?bid=<#.bid#>&refer=
    MMUSE_banners.sml-<#MAC.NBHOST#>";
    parent.backToTop=true; //blow away frame
 }
 window.defaultStatus=parent.defaultStatusMsg;
 parent.loadPage();
</SCRIPT>
```

13. Testing

The mousetrap *needs* to be tested thoroughly before it is put live. After the mousetrap has been completely setup, testing can start.

13.1. Get a Link To The Mousetrap

Put a test banner or link (with a test banner ID) that can be clicked on to jump to the mousetrap frameset page. Ideally the page with this link would be on another server. This link will be used to check if the referer is being passed properly through to the registration page. Also, by entering with the banner ID, we can verify that the banner ID is being passed to the registration page.

As an example, the following links were setup to test development sites in a computer named KAUAI:

```
<a href="http://kauai.sbusiness.com/xpics/ass/mass_index.sml?acb=acb100002-60000">Ass Award</a><br>

<a href="http://kauai.sbusiness.com/sexmuseum/mmuse_index.sml?acb=acb100002-60000">Sex Museum</a><br>

<a href="http://kauai.sbusiness.com/sexroulette/mroul_index.sml?acb=acb100002-60000">Sex Roulette</a><br>
```

13.2. Test All Links

Every link on every page needs to be tested. Make sure that the frameset is blown away (and without generating a new window) where the link has **target="_top"** in it.

Be sure to check the links on one of the footers, the banners page, registration pages, sales pages (including the member area and webmaster area).

In other words, check all links on every page!

13.3. Test Back & Warp on All Pages

Go to *each* page and hit the back button to verify that the "parent.backTo" gets loaded. Go to *each* page and warp out. You can do that my simply re-loading the frameset page. When you warp, you should open a new browser window with the proper page loaded in it. To test the next warp, you have to close the new window.

To test the softsales page back/warp, you will have to load the frameset page both with an ACB banner ID and without one. If you arrive with a banner ID, you will back/warp from softsales to our banners page. Without a banner ID, you will bypass our banners page and go to a GP rated type site (like CyberThrill).

13.4. Verifying That Parameters Are Passed

Using the link created in section 13.1, click to the frameset page (this creates a referer). Click the Enter button on the on the warning page and from the sales page, click to the first registration page.

Using your browser to view source (view frame source) and you will see the source of the registration page without the SML converted to HTML.

Scroll down the registration page and located the hidden input fields with the names of PARM6, PARM18, and PARM22 as shown below:

```
<input type="hidden" name="PARM6" value="http://kauai.sbusiness.com/jump.htm">
<input type="hidden" name="PARM18" value=" MMUSE_sales1_Live-xpics_server6">
<input type="hidden" name="PARM19" value="NEW">
<input type="hidden" name="PARM22" value="acb100002-60000">
```

PARM6 is the referer. In this case, we had the links to the frameset page (mentioned in section 13.1) on the domain of "kauai.sbusiness.com".

PARM18 is the feeder. The feeder should start with capital letters. If there are multiple links on the sales page, the feeder should indicate which link we came from. For example, the above feeder came from the "mmuse_sales1.sml" page with a click on the "Live Sex Shows". The page was served up from "xpics_server6".

PARM22 is the banner ID passed to the frameset page when it was first loaded.

These above three parameters must be present and functional.

13.5. Register Somebody

Using a test credit card, register a fictitious person (ask Gene to set up a test credit card). Register somebody once from *every* registration page, selecting different memberships. Put your sbusiness email address in the registration page so we can locate all test memberships and easily delete them. Using the "XPics - Adult City SuperServer Administrator" at:

http://admin.xpics.com/xpl/admin/

Verify that the username you choose is available. Then after retrieving that user's info, verify the following is correct:

1. Name
2. Feeder
3. Referrer
4. Membership type
5. Banner ID
6. Credit card & expiration

14. Troubleshooting

There are various simple troubleshooting tools and techniques available to quickly isolate a problem.

14.1. PARM6, 18, or 22 Missing

Make sure that you have a proper banner or link to the frameset page. When the frameset page is loaded from your link, you should see something like the following in the browser's location window:

http://www.sexmuseum.com/mmuse_index.sml?acb=acb100002-60000

You need to have the "sml?acb=acb100002-60000" parameter appended to the URL of the frameset page.

14.1.1. Check The Warning Page

With the warning page in the main frame (and using Netscape 4) right click anywhere on the warning page and select "View Frame Info" You should see something like the following:

http://www.sexmuseum.com/mmuse_warning.sml?SID=0380ed760164&bid=acb100002-60000

You can see that we are looking at info for the warning page on Sex Museum. There should be a number after the "SID=". Then the "bid=" should be following by the test banner ID. A problem here indicates the frameset definition for the "main" frame (on the frameset page) is not correct.

14.1.2. Check the Sales Page URL

Click on the Enter button and load the sales page. After the sales page is loaded right click anywhere on the warning page (using Netscape 4), and select "View Frame Info". You should see something like the following:

http://www.sexmuseum.com/mmuse_warning.sml?SID=0380ed760164&bid=acb100002-60000

As in the warning page, you should see a number after the "SID=" and the banner ID should be after the "bid=". If this is incorrect, there is a problem on the "Enter" link on the warning page (probably SML).

14.1.3. Check the Sales Page Links

With the sales page in the browser, view the source for the sales page (view frame source) and look for the links to the registration pages. Verify that these links are correct. For example, in Sex Museum, the link to the "Live Shows" looks like the following:

```
<A HREF="mmuse_livesex.sml?feeder=MMUSE_sales_1_Live-xpics_server6&SID=0380ed760164&bid=
acb100002-60000" OnClick="parent.clicked(this)" OnMouseOver="return parent.setStatus('Check It Out -
Live!')" OnMouseOut="return parent.setStatus('')">
```

Verify that the "feeder=" and "SID=" are present and are the correct format. Also the OnClick="parent.clicked(this)" needs to be present. OnMouseOver and OnMouseOut are optional.

14.1.4. Check the Registration Page

If the above all checkout, the problem has to be in the registration page. Verify in the registration file the following is correct:

```
<input type="hidden" name="PARM6" value="<#SID.referer#><#SID.tracked#>">
<input type="hidden" name="PARM18" value="<#.feeder#>">
<input type="hidden" name="PARM19" value="NEW">
<input type="hidden" name="PARM22" value="<#.bid#>">
```

14.2. Back or Warp Is Doesn't Work

As a trouble shooting aid, go the frameset page and the frameset definition itself near the bottom of the page. You should see the following:

```
<FRAMESET ROWS="100%,*" SCROLLING=no BORDER=0 frameborder=no framespacing=0
onUnload="unload()">
```

Change the frameset definition to the following:

```
<FRAMESET ROWS="*,40" SCROLLING=no BORDER=0 frameborder=no framespacing=0
onUnload="unload()">
```

After re-loading the frameset page, you will see a 40 pixel window at the bottom of the screen. It shows the hidden2 file which has the "backTo" and "warpTo" variables. You can verify that these variables are correct for each page. For example, from the warning page, these variables should be set for the softsales page.

If the mousetrap mechanism is working properly, the small window at the bottom (the hidden frame) should change from the hidden1 page to the hidden2 page with each link click and browser back button press. If it doesn't, check the JavaScript between the "</title>" and "</head>" and the "OnClick=..." JavaScript on suspected link.

15. Esoterica

15.1. Hosts File

When editing on a server that has multiple mirrors, the only way to guarantee getting to the proper is to enter the IP address instead of a URL. For example, if you would like to go to Ass on "xpics_server1", you would enter the following URL:

http://208.215.61.5/ass/mass_index.sml?acb=acb100002-60000

Assuming you have Windows NT, an alternative to the above is to open up the "hosts" file. On Windows NT 4, you should open the following file in Notepad (if you can't find the file there, just use "Find" off the "Start" button to locate the "hosts" file):

winnt\system32\drivers\etc\hosts

With the following entries in your hosts file, every time your browser requests "www.xpics.com", for example, you will automatically get directed to the 208.215.61.5 IP address.

```
127.0.0.1       localhost
208.215.61.5    www.xpics.com
208.215.61.12   www.sexmuseum.com
208.215.61.11   members.sexroulette.com
208.223.222.18  www.videosexchannels.com
208.223.222.31 www.xxxsexphotos.com
```

15.2. Starting the Mousetrap On a Non-Warning Page

The Mousetrap normally starts on the warning page. As discussed in section 5.1.1, the warning page filename is set in the frameset page.

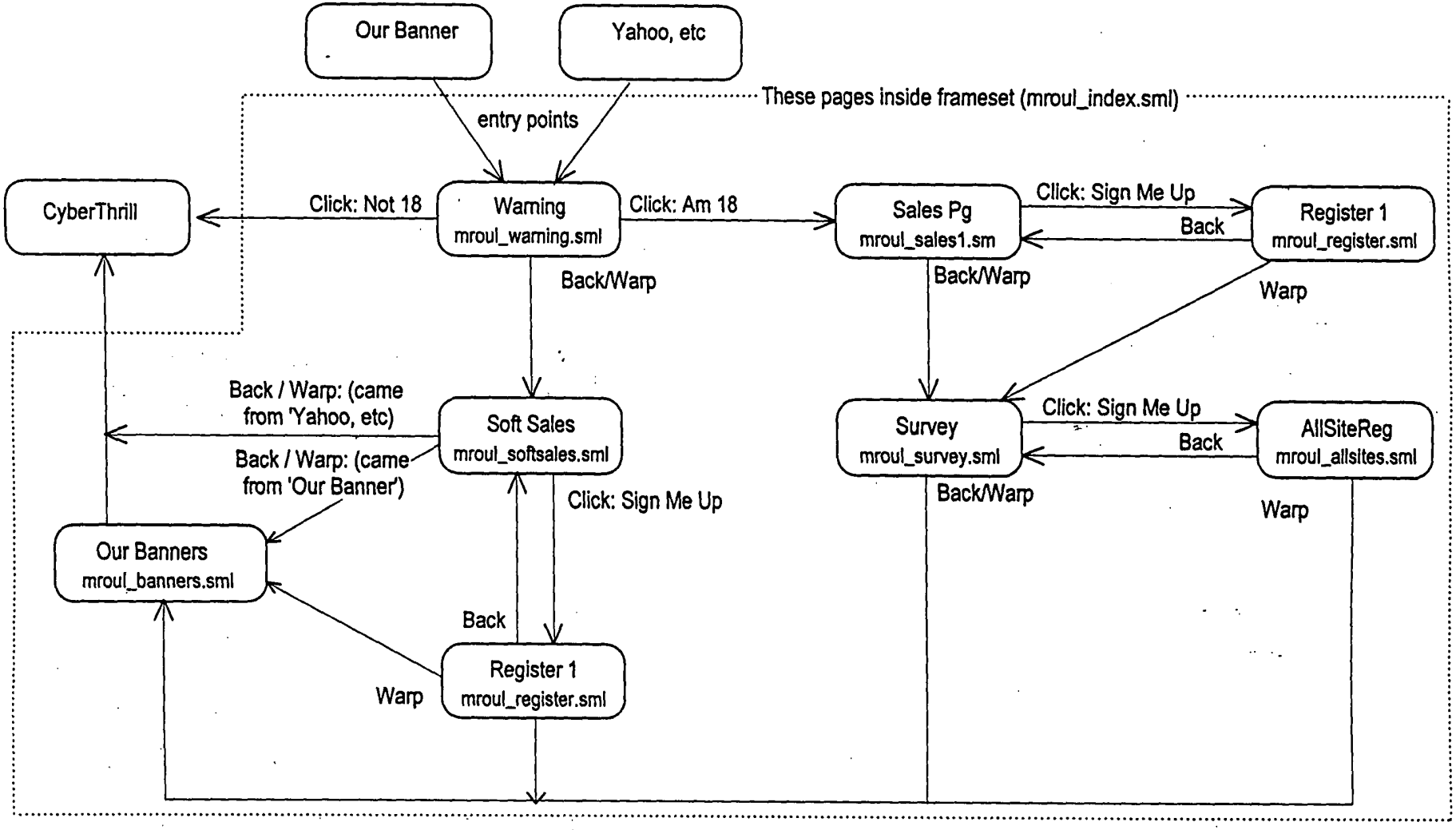
If you would like to start a mousetrap from at the sales page for example, you would enter the following URL:

http://www.sexmuseum.com/mmuse_index.sml?acb=acb100002-60000&startAt=mmuse_sales1.sml

This would start the mousetrap at the sales page in the Sex Museum mousetrap instead of the warning page.

This might be used when leaving backing/warping from the banners page and "enterClicked" is true (see section 12.2).

16. Mousetrap Flow Drawing -- Flow Chart for Sex Roulette



Sex Museum and Ass Awards are similar except there are 4 registration pages off the Sales Pages.

The Mousetrap Programmer's Guide

The Mousetrap
Programmer's Guide

1. The Mousetrap Effect

Normally the back button in a browser takes the user to the previous page. The "mousetrap" however, has the ability to redirect a user to a new URL when the back button is pushed. In addition if the user "warps" (change location via bookmarks, history list or typing in a new URL), a new browser window opens.

Lets look at a few senarios:

1. The user comes to the "Warning" page and hits the back button. They do not return to their previous URL but instead see a "soft" sales page (the photos are softcore). If they hit the back button again, they end up on a banners page. If they now decide to "warp" by going to their bookmarks, they will go the location they "warp" to but they will also open up a new browser which will overlay the warped-to browser window with yet another sales page.

2. The user comes to the "Warning" page and hits the enter button to go to a sales page. From the sales page, they click on a link and enter an age verification page. They do not have a credit card and decide to "warp" to a new location. Their browser opens up a new window with a "survey" page on it. They close the window with the survey and discover that the placed they warped to is now displaying in their original browser window.

3. The user comes to the "Warning" page and hits the enter button to go to a sales page. From the sales page, they hit the back button but instead of going back to the warning page, they see a "survey" page. If the user fills out the survey, they will get a free subscription to all sites. They fill out the survey and see the age verification page registration page. They fill out the form and enter the members area.

2. Operation Overview

The core function of the mousetrap is achieved through the use of a hidden frame and JavaScript. All pages in the mousetrap are displayed within a frameset which has two frames, a frame named "main" which fills up 100% of the frameset and a frame named "hidden" which is indeed hidden.

When the user clicks on a link (like the "Enter" on a warning page), the main frame (no pun intended) is loaded. While the browser is loading the main frame (with a sales page), it starts to execute the JavaScript on the sales page. This JavaScript runs the "loadPage()" function which then loads into the hidden frame, the "hidden1" page. The "hidden1" page then executes its JavaScript which then loads the "hidden2" page.

NOTE: To install a Mousetrap, you *must* read "The Mousetrap for HTML Designers".

This is the sequence of events for a link click:

Link Click Events (Click Enter on Warning Page) [1]

Event#	Description	Frameset	JavaScript[2]
1	Click Enter on Warning page	main	parent.clicked()
2	Load Sales1 Page	main	parent.loadPage()
3	Load Hidden1	hidden	loadPage() (in hidden1)
4	Load Hidden2	hidden	—

Notice that the last page loaded in the browser's history list is the "hidden2" page (hidden1 and hidden2 are frameset variables which hold the URL's of the documents loaded into the "hidden" frame).

When the user presses the browser's "back" button the hidden2 page is unloaded and the hidden1 page is loaded. The hidden1 page runs it's "loadPage()" function which then loads a page into the "main" frame. The user will not return to the warning page but instead will see a second sales page in the "main" frame. The Sales2 page will then load the Hidden2 page into the hidden frame:

This is the sequence of events caused by pressing the browser's back button:

Back Button Event (Backing from First Sales Page)[1]

Event#	Description	Frameset	JavaScript[2]
1	Browser's back button clicked	—	—
2	Load Hidden1	hidden	loadPage() (in hidden1)
3	Load Sales2 Page	main	parent.loadPage()
4	Load Hidden2	hidden	—

From the second sales page, if the user decides to "warp" by going to Yahoo via a bookmark, the frameset page itself will unload. When the frameset unloads, its "onUnload" event causes the execution of its "unload()" function. This function creates a new window with yet another sales page in it (perhaps a page of banners).

This is the sequence of events caused by "warping" to Yahoo:

Jumping to new URL via Bookmark[1]

Event#	Description	Frameset	JavaScript[2]
1	Original window loads Yahoo	main	frameset's "unload()"
2	New Window loads hidden1 page and banners page	hidden main	— parent.loadPage()
3	Load Hidden2	hidden	—

Note 1: At the start of events, the "hidden2" page is currently in the "hidden" frame.
Note 2: This is the JavaScript function which initiates the next event.

3. The Details (Overview)

The next three sections explain the functions and interactions of the frameset, main, and hidden pages. See Appendix A for flowchart drawing of Sex Roulette sales site (as of February 9, 1998).

This section assumes a familiarity with SML (see *SML for Dummies* at the following URL: http://kona.sbusiness.com/sml_for_dummies/). This section also assumes familiarity with JavaScript.

Note: The mousetrap mechanism allows the normal use of sales pages if the browser has JavaScript turned off or is not JavaScript enabled. It is even possible to access the sales pages if the browser is not "frames" enabled. In these cases, the back button will operate normally.

Netscape 2 executes the back button at the frameset level and removes the entire frameset. This means that the "back" button acts like a "warp" in Netscape 2.

4. The Frameset Page

The frameset page contains the JavaScript functions and variables accessed by the various pages loaded into the "main" and "hidden" frames. The beauty of frameset level variables, is that state can be maintained between pages. Typically this page is named something like "mmuse_index.sml" or "mass_index.sml" (the first "m" stands for mousetrap).

We will go through the frameset page line by line and provide comments.

4.1. The .goTo and .startAt SML variables

Normally the warning page is first loaded when the mousetrap starts up. However, the following URL would allow the starting at another page:

http://www.sexmuseum.com/?startAt=mmuse_sales1.sml

The first of the following lines allows that to happen (they must be in this order):

```
<$if cond:.startAt,cond:!.goTo val:.goTo=.startAt><$endif>
<$if cond:!.goTo val:.goTo="mmuse_warning.sml"><$endif>
```

It might be easiest to examine the overall effect of the these two lines by looking at the "goTo" defined condition:

Condition1 - goTo not defined: If "startAt" is defined, "goTo" is assigned to it. Otherwise the warning page is loaded.

Condition2 - goTo defined: When the user warps, a new window is opened with a URL that may look like the following:

http://www.sexmuseum.com/?bid=acb100002-60000&goTo=mmuse_survey.sml

The "goTo" variable is used to in the frameset definition to load a page into the main frame.

4.2. The .acb SML variable

Typically the sales sites are entered through the clicking of a banner. That banner includes in its URL a parameter to identify the person and banner. The following URL will load the sex museum sales page:

http://www.sexmuseum.com/?acb=acb100001-g1400

In the "acb" parameter (ad campaign banner), the 100001 is the advertiser number and the g1400 is the number corresponding to a particular banner.

Raw hits are assigned to an advertiser whenever the server notices a page loaded with the "acb" parameter in the URL. With the following line:

```
<$if cond:.acb val:.bid=.acb><$endif>
```

If the .acb variable exists (not everyone comes from a banner), the SML variable .bid is assigned to its value. On subsequent pages, the advertiser number is passed by appending the following to the URL:

?bid=acb100001-g1400

We don't want to append "?acb=acb100001-g1400" because this could inadvertently cause another raw hit to be assigned to the advertiser.

4.3. Constants

4.3.1. defaultStatusMsg

Each page sets the default message by loading the following variable:

```
defaultStatusMsg="Welcome to Sex Museum";
```

4.3.2. Hidden Frame Pages

The hidden frame gets loaded with the URLS of the following variables.

```
hidden1="mmuse_hidden1.sml"; //Change in frameset definition too
hidden2="mmuse_hidden2.sml";
```

Note that the URL at hidden1 must also be set in the frameset definition.

4.3.3. defaultWarpTo

It appears that Netscape 2 cannot retain variables in its parent frame when going from page to page.

```
defaultWarpTo="mmuse_survey.sml";//Kludge for Netscape 2 - warp
location in registration off sales 1
```

The registration (age verification) pages can be reached from different sales pages (the softsales and regular sales pages may use the same registration pages). Because of this, the registration pages do not re-define the "warpTo" or "backTo" variables and do not run the parent.loadpage() function as well. This allows a normal "back" browser function to occurs and the user can return to the previous sales page.

With Netscape 2, if someone warps out of a registration page, the parent.warpTo variable contains garbage. If there is garbage in the "warpTo" variable, the "defaultWarpTo" value is used instead. This variable is used in the "unload()" function (see section 0).

4.4. Variables

Each page loaded into the "main" frame sets the window's "defaultStatus with the following variable:

4.4.1. window.defaultStatus

This sets the default status message for the current window, which is the frameset page.

4.4.2. acb

The following loads the "acb" variable with the current banner ID:

```
acb ="<#.bid#>";
```

This variable is checked in the softsales page to decide whether to send the user to our banners page. If the user has come from Yahoo, for example, they won't have a banner ID and they may also be under age so they are not sent our banners page.

4.4.3. backToTop

When the user hits the back button, they normally load a new page within the current frameset. The following line sets this default:

```
backToTop=false;
```

In the "banners" page, however, this variable is set to true (parent.backToTop=true). This allows the user to back out of the banners page and totally exit the mousetrap: the frameset page gets unloaded and replaced with a new page (the mousetrap's last gasp).

4.4.4. linkClicked=false

When a linked is clicked this variable is set to true.

The sequence of loading the main frame and the hidden frame pages varies, depending on the setting of this variable. This variable is set in the "clicked(aLink)" function and used in a function defined in this section ("loadPage()") and in the hidden1 page.

4.4.5. warpTo=""

This variable is set by the various pages loaded into the main frame. If the user tries to leave the sales pages via bookmarks, history list, or just typing in a URL, the user will end at the URL/file pointed to by this variable.

4.4.6. newWin=true

The setting of this variable determines if we should create a new window at warp time. This is normally set to true except when the submit button is clicked on the registration page. After the submit button is pressed, the user may get presented with a page that has a link to the members area. This link has the 'target="_top"' attribute and we don't want to create a new sales window when the frameset is blown away as the user tries to enter the members area.

4.4.7. timerID=false

This variable is used in the "setStatus(msg)" function to indicate if a timer is running.

4.4.8. firstPageLoaded=false

This variable is used and set in the hidden1 page. Normally hidden1 would load hidden2 but the first time hidden1 is loaded, hidden1 does nothing because this variable is set to false. This allows the first page loaded into the main frame to load hidden2 and thus guaranteeing that hidden2 will load *after* the first main frame page.

4.4.9. enterClicked=false

This variable is currently used on the Sex Museum site. It is set to true on the "mmuse_survey.sml" page. If the user gets to the survey page and then hits the back button, they will end up on the banners page. It is also possible to get to the banners page by hitting the back button from the warning page. For users who have gotten to the banners page and have clicked "Enter" on the warning page, we drop them into the sales pages of another site.

4.4.10. initialLocation

The following initializes the initialLocation variable:

```
i=location.href.indexOf("?acb=");
if (i==-1) initialLocation = location.href;
else initialLocation = location.href.substring(0,i) + "?bid" +
   location.href.substring(i+4,location.href.length);
```

This variable is used in the "unload()" (see section 0). Basically the above lines change the "acb" parameter name in a URL to "bid" if the "acb" exists.

For example:

http://www.sexmuseum.com/mmuse_index.sml?acb=acb100002-60000

becomes:

http://www.sexmuseum.com/mmuse_index.sml?bid=acb100002-60000

If the user warps out and a new window with a new frameset is loaded, we need to change "acb" to "bid" because it might be possible to give another payed click to the same advertiser (the server looks for "acb=acb..." in a URL to assign a visit to an advertiser).

4.5. The Functions

4.5.1. function unload()

This function runs when the frameset page is unloaded caused by the user attempting to warp away from our sales site.

```
if ((warpTo.indexOf(".sml") == -1)&&(warpTo.indexOf(".htm")== -1)
   && (warpTo.indexOf("http") == -1)) warpTo = defaultWarpTo;
```

The above lines are used to get around a limitation in Netscape 2 JavaScript. Variables set in the frameset (parent) page are not retained when a new page is loaded into the main frame. See "defaultWarpTo" in section 4.3.3 for more information. A corrupted "warpTo" is detected by checking to see if it has the ".sml", ".htm", or "http" strings in it.

```
if ((window.name != "newWindow") && newWin)
```

A new window will only be created at warp time if we are not already in a new window and the "newWin" is true. We only create a new window at warp time only once so the user can leave the sales site the second time they warp out.

```
{
   if (backToTop)
      window.open(warpTo,'newWindow');
```

If "backToTop" is true, the frameset page will be replaced with the URL indicated by the "warpTo" variable and we will have exited the mousetrap.

```
else
{
    if(initialLocation.indexOf("?") == -1)
       initialLocation=initialLocation + "?";
   else initialLocation=initialLocation + "&";
   window.open(initialLocation + 'goTo=' + warpTo,'newWindow');
}
```

Next the function checks to see if "initialLocation" has a "?", if not it adds one, otherwise it adds a "&". Finally, we reload the frameset page with the "goTo" parameter appended to the end of the initial frameset's URL.

4.5.2. function clicked(aLink)

This function needs to run every time a link is clicked. Every link should invoke this method via the "onClick" event.

```
linkClicked=true;
if (aLink.target && (aLink.target.indexOf('_top') == 0))
   newWin = false;
```

This function sets "linkClicked" and then checks to see if the link's target attribute is set to "_top". If so, we don't want to create a new window when this link is clicked ("newWin" is used in the "unload()" function [see section 0]).

4.5.3. function setStatus(msg)

This function is called from within all pages that wish to set the status message when the user places the mouse pointer over a link.

```
top.main.status=msg;
if (timerID) clearTimeout(timerID);
if (msg != '') timerID=setTimeout("window.status=''",5000);
return true;
```

The timer function is used because Internet Explorer does not support a onMouseOut event. The timer will automatically clear the status message after 5 seconds. If the timer is running when this function is entered, it gets initially cleared before being set again.

4.5.4. function loadPage()

This function along with the function in hidden1 contain the logic to control the back button's action.

```
if (!firstPageLoaded) {setTimeout('loadPage()',500);return}
```

The warning page is the first page to call "loadPage()." If hidden1 has not been loaded (indicated by "firstPageLoaded" being false), the warning page cannot load hidden2. We want to be sure that hidden1 has been loaded before hidden2 is loaded (otherwise there will be no browser history of hidden1 preceding hidden2).

The above code starts a cycle of calling the "loadPage()" (i.e. calling itself) every ½ second until hidden1 is loaded. Once hidden1 is loaded (and it sets "firstPageLoaded to false), "loadPage()" can continue to the next section:

```
if(!parent.linkClicked)
   parent.hidden.location.href=hidden2 + "?<#SID=.#>";
else
   parent.hidden.location.href=hidden1 + "?<#SID=.#>&bid=<#.bid#>";
```

In the first case, "linkedClicked" is false. This means that the current page in the main frame (the one calling this function) was not loaded through a link click. That is it was was loaded by the "loadPage()" function in hidden1 (which means the browser's back button was pressed). Therefore the hidden2 must be loaded in the hidden frame. See "Operation in a Nutshell" page 1 for details in the sequencing of a browser's back button press.

In the second case, the current page in the main frame (the one calling this function) was loaded through a link click. This means hidden1 must be loaded in the hidden frame. Hidden1 will then load hidden2. See "Operation in a Nutshell" page 1 for details in the sequencing of a link click.

4.5.5. Frameset Definition

The frameset definition sets up a page with two frames, one taking up 100% of the frameset and the other taking up the remainder (which is nothing so the frame is hidden).

```
<FRAMESET ROWS="100%,*" SCROLLING=no BORDER=0 frameborder=no
framespacing=0 onUnload="unload()">

<FRAME SRC="<#.goTo#>?<#SID=.#>&bid=<#.bid#>" NAME="main"
SCROLLING=auto BORDER=0 frameborder="0">

<FRAME SRC="mmuse_hidden1.sml?<#SID=.#>&bid=<#.bid#>"
NAME="hidden" SCROLLING=no BORDER=0 frameborder="0">
```

The main frame gets its source set from the .goTo SML variable. When the frameset is first loaded, this variable is set to the warning page. If there is a warp, this variable is set to whatever the "warpTo" variable was set to at warp time. The SML state ID and the banner ID parameters are also passed in this URL.

The hidden frame initially loads the hidden1 file, passing to it the SML state ID and the banner ID parameters.

4.5.6. NOFRAME

This section allows a non-frames browser access to the sales pages. The .goTo will take the user to the warning page or the warped to page.

```
<NOFRAME><HTML><HEAD>
<TITLE>Sex Roulette</TITLE></HEAD>
<BODY TEXT="#FFB000" BGCOLOR="#000000" LINK="#DD0000"
     VLINK="#DD0000" ALINK="#FFFF80">
Click <a href=<#.goTo#>>here</a> to continue...
</BODY>
</HTML>
</NOFRAME>
```

5. The Hidden Frame

The hidden frame is the secret behind the operation of the mousetrap. The hidden frame always loads last. Consequently when the browser's back button is pressed, the hidden2 file first unloads and then the hidden1 file gets loaded. Hidden1 has a JavaScript which then loads a page into the main frame.

5.1. The hidden1 File

This file is loaded into the hidden frame and is used to load the main frame when the back button is pressed.

```
function loadPage()
{
  if (parent.firstPageLoaded)
  {
    if (!parent.linkClicked)
    {
       if (!parent.backToTop)
         parent.main.location=parent.backTo +
           "?<#SID=.#>&bid=<#.bid#>";
       else
       {
         parent.newWin=false;
         top.location=parent.backTo;
       }
    }
    else
      parent.hidden.location=parent.hidden2;
    parent.linkClicked=false;
  }
  else parent.firstPageLoaded = true;
}
```

5.1.1. if (parent.firstPageLoaded)

We first check to see if this is the first time this file is being loaded. If so we do nothing except set the "parent.firstPageLoaded" variable to true. The first time this file is being loaded is when the frameset is being first loaded... the frameset loads this page into the hidden frame. Normally hidden1 would load the main frame but initially, this is being done by the frameset page.

5.1.2. if (!parent.linkClicked)

We need to check if this page got loaded by the pressing of the back button or by the "parent.loadPage()" being executed by the page in the main frame.

If the "parent.linkClicked" variable is true, then this page was loaded by the main frame page after a link click and we need to load the hidden frame with the hidden2 file.

Otherwise this page got loaded by the hitting of the browser's back button and we need to do further testing.

5.1.3. if (!parent.backToTop)

If "backToTop" is false, we will load the main frame with the "backTo" page. Otherwise we will replace the entire frameset with the "backTo" page.

5.2. The hidden2 File

The logical purpose of this page is mostly that of a placeholder... something to load after the main frame and hidden1 is loaded. The JavaScript in the body of this page is used strictly for testing purposes:

```
<script language="JavaScript">
   document.write("parent.backTo=" + parent.backTo + " | | | | | | ");
   document.write("parent.warpTo=" + parent.warpTo  + " | | | | | | ");
</script>
```

To use the above in testing you have to modify the frameset page and the frameset definition itself near the bottom of the page. On the frameset page you should see the following:

<FRAMESET ROWS="100%,*" SCROLLING=no BORDER=0 frameborder=no framespacing=0 onUnload="unload()">

Change the frameset definition to the following for testing with the hidden2 file:

<FRAMESET ROWS="*,40" SCROLLING=no BORDER=0 frameborder=no framespacing=0 onUnload="unload()">

6. The Main Frame

The main frame contains all the pages the user views. Most pages have a similar JavaScript between the </title> and </head>.

6.1. The Warning Page

The warning page has the following JavaScript:

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Script start
   parent.backTo="mxsp_softsales.sml";
   parent.warpTo="mxsp_softsales.sml";
   window.defaultStatus=parent.defaultStatusMsg;
   parent.loadPage();
//  Script end -->
</SCRIPT>
```

The first two lines set the backTo and warpTo variables in the frameset page. The third line sets the default message. The fourth line starts the loadPage() function (which starts the sequence of loading into the hidden frame).

6.2. Sales Page

6.2.1. MAC.visit

The sales page has an SML tag on top of it:

```
<#MAC.visit#>
```

This SML tag causes a visit to be registered for the advertiser who's banner ID is appended to the URL as below:

http://www.xxxsexphotos.com/mxsp_index.sml?acb=acb100002-60000

When the user clicks the enter page and loads the page with the "MAC.visit" tag, the advertiser with the number "100002" gets paid for a click.

This tag should not be on the softsales page. We don't pay an advertiser when they attempt to back off the warning page.

The JavaScript for this page is identical to the warning page except for one additional line:

```
parent.enterClicked=true;
```

When this variable is true, the user has clicked the enter button on the warning page. The banners page may optionally look at this variable to decide where to redirect the user.

6.3. Registration (Age Verification) Page

The JavaScript on this page is much different that the sales/warning/banner pages. Also, the registration page may eventually change to a multi-page system currently under development (Feb 1998).

The JavaScript on the current (single) page registration page looks like the following:

```
<script language="JavaScript">
<!-- Script start
     var isSent = 0;
     parent.newWin=true;

     function load()
     {
        window.defaultStatus=parent.defaultStatusMsg;
     }

     function SubmitData( form )
     {
        parent.newWin=false;
        if( !isSent )
     {
        isSent = 1;
        form.submit( );
     }
    }
//  Script end -->
</script>
```

6.3.1. No loadPage()

The first thing to note is that the four lines on the warning page are missing here, including the "loadPage()" function. This is because if the user hits the back button, we want them to be able to return to the sales page. If they warp out, they will warp to the location set in the previous sales page (whether it was softsales or a regular sales page).

6.3.2. isSent

The var "isSent" is used to guarantee that the pressing of the submit button will only submit once, even if the user keeps pressing it.

6.3.3. Parent.newWin

In the "SubmitData" function, "parent.newWin" is set false so that after the person registers and clicks to go to enter the members are, they will not get a new sales page window when they warp. Should the person hit the back button from a failed registration page, the "parent.newWin=true" line will allow a new window if the person tries to warp from the registration page.

6.3.4. The Submit Button

The submit button, near the bottom of the page uses an SML conditional to decide whether to use a JavaScript "button" or a regular "submit" button. The variable "SID.js" is set to true in the "hidden2" page. This page could *only* be loaded if the user had JavaScript capable browser with JavaScript enabled. Therefore the code below will only us a JavaScript button if the user is running JavaScript:

```
<$if cond:SID.js>
<CENTER><input type="button" value="Submit Information"
onClick="SubmitData( this.form )"></CENTER>
<$else>
<CENTER><input type="submit" value="Submit Information"></CENTER>
<$endif>
```

The SML conditional <$if cond:MAC.user_agent LIKE "Mozilla/[2-9]"> should only be used on non-mousetrap registration pages. We have found about 5% of users have turned off JavaScript on a JavaScript capable browser. The condition searching for "Mozilla[2-9]" will present those users with a dead button and they will not be able to register.

6.4. Survey Page

This page currently takes a survey of users but never stores the data. In development is a survey page and Perl script which will store the data to SQL Server (this project was complete in early Feb, 1998).

This page combines the JavaScript of both the warning page (section 6.1) and the registration page (section 6.3). The submit button can be a JavaScript button because the user must have JavaScript to get to this page.

The fields in this form are all passed as parameters to the URL of the "allsites" registration page. The hidden fields are used to pass the "referrer", the "bid", and the "referer". Because this is a submission, we have found it is possible to end up on a different server for the "allsites" page (especially when a Perl script "redirect" is eventually used to store the survey data). For this reason, the SID is useless and we have to directly pass the referrer as a parameter (the SID is specific to a server).

6.5. Allsites Registration Page

The JavaScript in this page does not refer to the parent. We found that IE4 generated JavaScript errors when the JavaScript referred to parent variables and functions after a form submission.

The JavaScript for this page is identical to the JavaScript in a non-mousetrap registration page. The submit button can be a JavaScript button because the user must have JavaScript to get to this page.

This page uses multiple SML conditionals to answer some of the issues raised in the survey page. For example, if the person is concerned about security, a paragraph will appear on this page that discusses security.

This page also has a certificate on it that uses an SML conditional to detect Navigator 3. This is because Navigator 3 cannot do cell background images so for Navigator 3, we had to chop the image up. IE3, IE4, and Navigator 4 can do cell background images and this system works better (the background pops up all at once).

6.6. Softsales Page

The JavaScript in this page is a little different than a regular sales page. For one, we are not setting the "parent.enterClicked" variable because the person did not click the enter on the warning page to get here (they hit the back button or warped from the warning page).

Also this page has a JavaScript conditional. If the person came in with a banner ID (variable "acb" is set on the frameset page... see section 4.4.2), they will back/warp to our banners. Otherwise they will go to a PG rated type website.

```
<script language="JavaScript">
<!-- Script start
   if (parent.acb == "")
   {
      parent.backTo=
        "http://198.168.54.139/thrill-cgi/ad/xpics/ricochet.cgi?xpicsr";
      parent.warpTo=
        "http://198.168.54.139/thrill-cgi/ad/xpics/ricochet.cgi?xpicsr";
      parent.backToTop=true; //blow away frame
   }
   else
   {
      parent.backTo="mxsp_banners.sml";
      parent.warpTo="mxsp_banners.sml";
   }
   window.defaultStatus=parent.defaultStatusMsg;
   parent.loadPage();
//  Script end -->
</script>
```

6.7. Banners Page

This page contains banners which promote Xpics' other sites. It may also contain banners of non-Xpics' sites.

6.7.1. Links

The banner links in this page all need the 'target="_top"' and the 'OnClick="parent.clicked(this)"' statements in all links. A sample link is shown below:

```
<a href="http://www.xpics.com/ass/ass.html?acb=acb100001-1100"
   Target="_top" OnClick="parent.clicked(this)">
```

6.7.2. Unconditional Back/Warp

This page has a line of JavaScript no used on previous pages:

```
<SCRIPT LANGUAGE="JavaScript">
   parent.backTo=
     "http://198.168.54.139/thrill-cgi/ad/xpics/ricochet.cgi?xpicsr";
   parent.warpTo=
     "http://198.168.54.139/thrill-cgi/ad/xpics/ricochet.cgi?xpicsr";
   parent.backToTop=true; //blow away frame
   window.defaultStatus=parent.defaultStatusMsg;
   parent.loadPage();
</SCRIPT>
```

The line "parent.backToTop=true" allows us to replace the frameset page with the pages at the backTo/warpTo URL's. This means the user will be exiting the mousetrap and is free to warp to wherever they wish. Of course, if they hit the back button again, they will be back to the mousetrap.

6.7.3. Conditional Back/Warp

This page can optionally have a conditional in the JavaScript. If the person gets to this page via the Survey page, they have already clicked the enter button on the warning page. This means that we can drop them to a sales page on another Xpics site when they try to back/warp. The JavaScript for this page follows:

```
<SCRIPT LANGUAGE="JavaScript">
   if (!parent.enterClicked)
   {
     parent.backTo=
       "http://198.168.54.139/thrill-cgi/ad/xpics/ricochet.cgi?xpicsr";
     parent.warpTo=
       "http://198.168.54.139/thrill-cgi/ad/xpics/ricochet.cgi?xpicsr";
   }
   else
   {
     parent.backTo="http://www.videosexchannels.com/vsc_sales.sml?bid=
       <#.bid#>&refer=MMUSE_banners.sml-<#MAC.NBHOST#>";
     parent.warpTo="http://www.videosexchannels.com/vsc_sales.sml?bid=
       <#.bid#>&refer=MMUSE_banners.sml-<#MAC.NBHOST#>";
   }
   parent.backToTop=true; //blow away frame
   window.defaultStatus=parent.defaultStatusMsg;
   parent.loadPage();
</SCRIPT>
```

Notice in the JavaScript, the "refer" variable is being passed to the "vsc_sales.sml" page. This accomplishes two things:

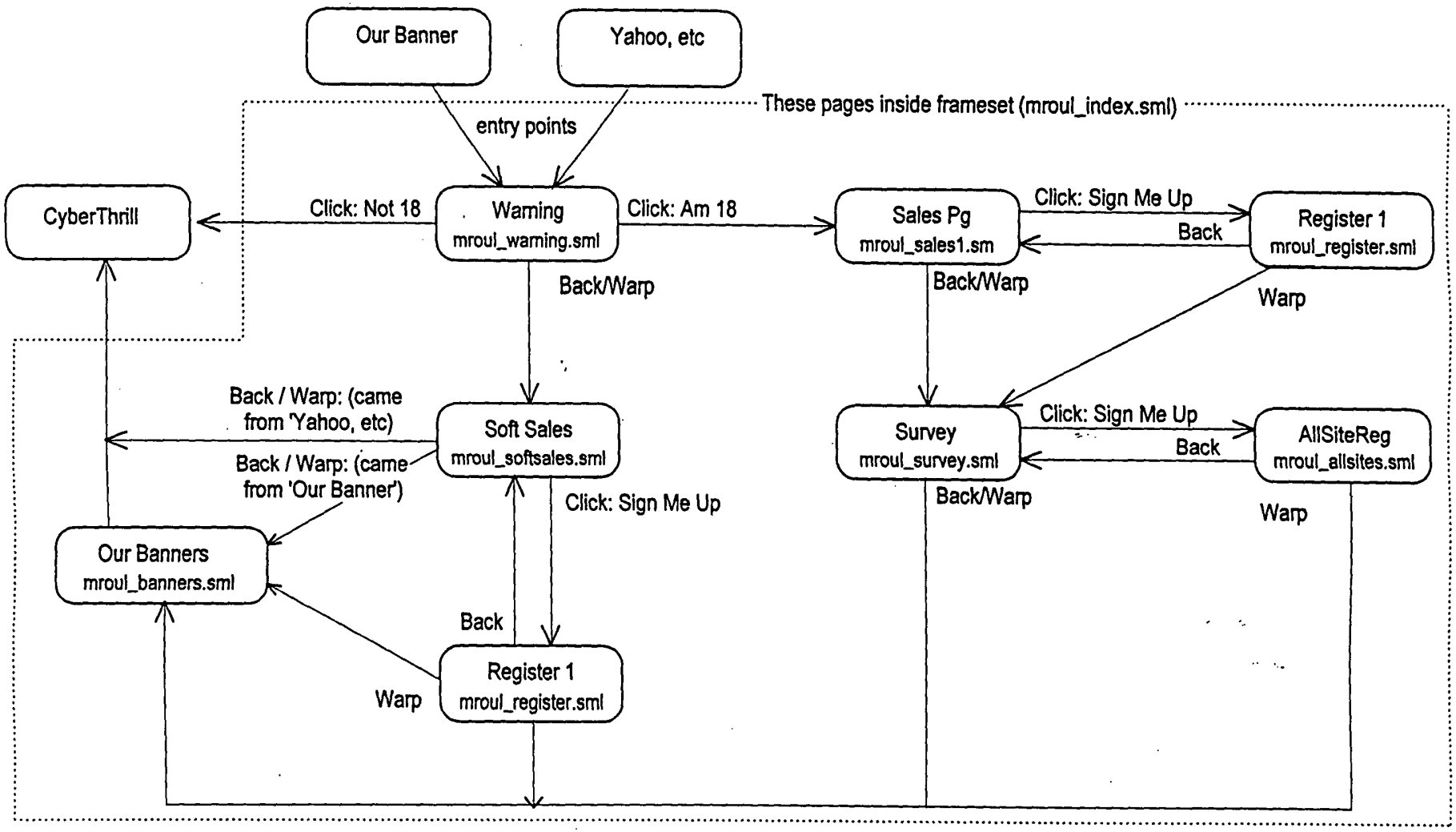
1. There will be a referrer in VSC that will indicate the user came from our own banners page (via back/warp of course)
2. The SML "MAC.visit" will not be used on the sales page.

The top of the "vsc_sales.sml" page should then have the following conditional:

```
<$if cond:.refer val:SID.referer=.refer>
<$else>
<#MAC.visit#>
<!-- VH -->
<$endif>
```

On the "vsc_sales.sml" page, we don't want to cause a visit to be registered to a banner ID if the user is jumping to it via the banners page.

7. Mousetrap Flow Drawing -- Flow Chart for Sex Roulette



Sex Museum and Ass Awards are similar except there are 4 registration pages off the Sales Pages.

Survey Page With SQL Server Access

1. Introduction

The survey page uses a Perl Script with to store information in an SQL Server database table. The form on the server page has numerous hidden variables which describe which parameters are stored and which are required to even cause a store.

This document assumes an understanding of "The Mousetrap for Programmers" document.

2. Survey Page JavaScript

The JavaScript for this survey page is identical to a survey page that does not store data. The following is located between them </title> and the </head> HTML marks:

```
<script language="JavaScript">
<!-- Script start
   parent.backTo="mass_banners.sml";
   parent.warpTo="mass_banners.sml";
   window.defaultStatus=parent.defaultStatusMsg;
   parent.newWin=true;
   parent.loadPage();
   var isSent = 0;

     function SubmitData( form )
     {
       parent.newWin=false;
       if( !isSent )
       {
         isSent = 1;
         form.submit( );
       }
     }
//  Script end -->
</script>
```

Also note that the submit button can be a JavaScript button because the user must have JavaScript to get to this page.

3. Survey Page Form

The hidden fields of the form configure this form to work with a particular table in a database (the database is selected by the DSN and the DSN is selected in the Perl script).

The following is the configuration section of the form:

```
<form name="qForm" method=get action="/cgi-bin/survey.pl">
  <input type=hidden name="SID" value="<#SID.#>">
  <input type=hidden name="referer" value="<#SID.referer#>">
  <input type=hidden name="js" value="<#SID.js#>">
  <input type=hidden name="bid" value="<#.bid#>">
  <input type=hidden name="feeder" value="MMUSE_survey-<#MAC.NBHOST#>">
  <input type=hidden name="table_name" value="survey_response">
  <input type=hidden name="redirect" value=" http://www.sexmuseum.com/mmuse_allsites.sml">
  <input type=hidden name="redirect_append" value="q1,feeder,referer,bid,SID,js">
  <input type=hidden name="one_required" value="q1,q1_text,q2,q2_text,q3_text">
  <input type=hidden name="stored" value="q1,q1_text,q2,q2_text,q3_text,feeder">
```

The first line calls a get to "survey.pl" when a submission occurs. The "SID", "feeder", "bid", "SID.js", and "SID.referer" fields get passed to the registration page .

The "feeder" line shows that the user came from the survey page in Sex Museum. This is the only line in the form that must be customized for a particular site.

The "table_name" parameter is the name of the table in SQL_Server that will receive the data. The "redirect" is the next page to load after the Perl script runs.

The "redirect_append" parameter indicates which parameters should get append to the "redirect" URL. Currently we need to append "q1" the "...allsites.sml" page answers questions, depending on what the selections are to the first question in the survey. If we wanted to respond to other questions, we would have to add that in the "redirect_append" list. Note that each parameter is separated by a comma.

The "one_required" parameter is used to indicate that at least one of these parameters needs to be set in order for us to store a record in the table. We don't want to store information if the user has not made any entries in the form.

The "stored" parameter is the names of the parameters we want to store. These names need to be exactly match the names in the table on SQL Server.

4. Survey Page Perl

4.1. Description

The Perl is fairly well commented so it requires little explanation here. The ODBC module must be installed on the Perl of every server that uses this Perl script. We tested this with Perl 5.001.

If there are any errors storing into a table, the Perl script displays a diagnostic error page.

4.2. Installing the ODBC Module

The "perl\lib\win32" directory needs to have the "odbc.pm" file installed. Also the "perl\lib\auto\win32\odbc" directory needs to be created and in that directory, the "odbc.pll" needs to be placed. These extensions were developed by Dave Roth and further info can be found at the following URLS:

```
http://multiweb.lib.calpoly.edu/odbc/
http://www.roth.net/odbc/odbcfaq.htm
http://citel2.tp.ac.sg/stfindex/staff/kinchew/odbc/odbcfaq.htm
http://www.iftech.com/oltc/webdev/webdev4.stm
http://www.ptyx.com/howto/index1.html
```

The above URL's are given for reference only... it is not necessary to read the above documentation to make the survey operate. The ODBC Perl files can be downloaded at:

ftp://ftp.epix.net/pub/languages/perl/authors/Dave_Roth/Win32odbc_v970208.zip

The above file has further documentation on the ODBC modules.

4.3. The MS SQL Server Table

Before the Perl can send data to SQL Server, a table with the name "survey_response" needs to be setup like the following (such a test table exists on PALERMO in the "sales_info" database):

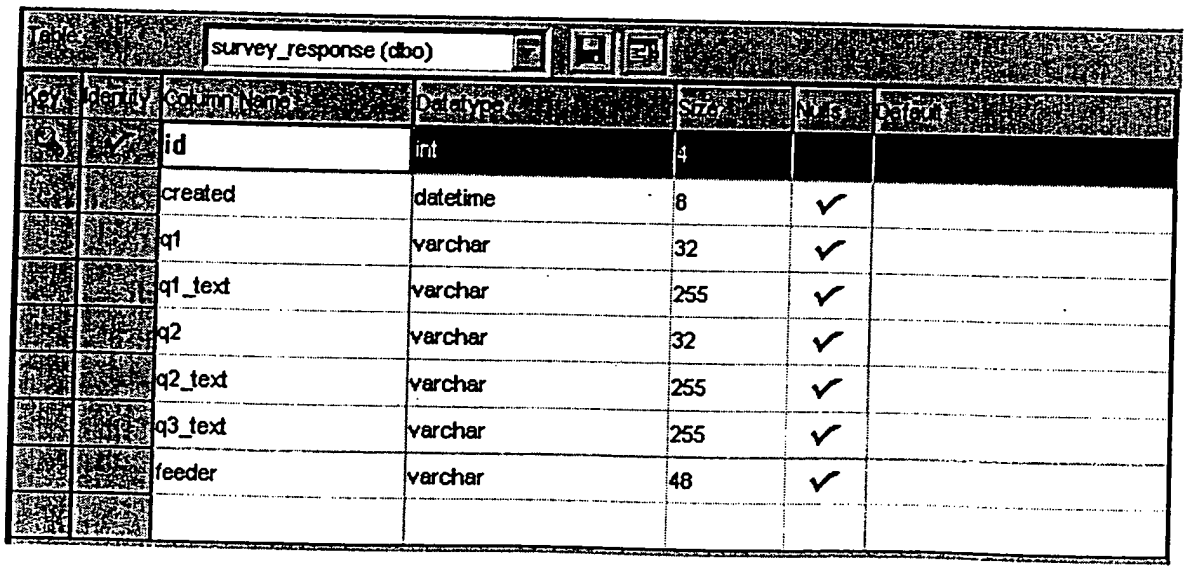


Table: survey_response (dbo)

Key	Identity	Column Name	Datatype	Size	Nulls	Default
🔑		id	int	4		
		created	datetime	8	✓	
		q1	varchar	32	✓	
		q1_text	varchar	255	✓	
		q2	varchar	32	✓	
		q2_text	varchar	255	✓	
		q3_text	varchar	255	✓	
		feeder	varchar	48	✓	

4.4. The Perl Script

The $dsn, $uid, and $pwd variables in the script need to entered after an SQL Server database and DSN is setup.

```
##############################################################################
# SURVEY ver 1.0  Copyright 1998 SuperBusiness Net                           #
# Written by Wayne W. Weber January 26, 1998  Last Modified: January 26,1998 #
#                                                                            #
# Inserts Values from an HTML form into a Database via ODBC                  #
#                                                                            #
##############################################################################
$dsn="PALERMO";
$uid="sa";
$pwd="xxxxxxx";

require "cgi-lib.pl";
use Win32::ODBC;

################ START OF MAIN ################

#### Parse arguments
&ReadParse(*in);

#### Flush StdOut
$| = 1;

#### Get arrays of fields that aren't required or stored
@one_required=split(/ *, */,$in{"one_required"});  #At least one of these fields must have
                                                   #something to store in DB
@stored=split(/ *, */,$in{"stored"});                    #Don't store these params in DB
@redirect_append=split(/ *, */,$in{"redirect_append"});  #Append params to redirect URL

#### See if any of the one_required fields entered ####
$one_req_found=0;
foreach $req (@one_required)
{
   if ($in{$req} ne "") {$one_req_found=1;}
}

#### Construct the date/time value ####
($sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdat) = localtime(time);
++$mon; $date_time = "$mon-$mday-$year $hour:$min:$sec";

#### Find fields to be stored ####
foreach $store (@stored)
{
   if ($in{$store} ne "")
   {
      $insert{$store}=$in{$store};
   }
}

#### Execute the SQL Insert command subroutine ####
if ($one_req_found)   #Don't insert a record if nothing entered
{
   $error_found="";
   if (($error_found = &insert_sql($in{'table_name'},\%insert,$date_time,$dsn,$pwd,$uid)) ne "")
   {
      &print_error("SQL Error",$error_found);
   }
}
```

```
    #### Append parameters indicated in @redirect_append ####
    $append="";
    foreach $element (@redirect_append)
    {
       if ($in{$element} eq "") {next;}          #Don't append if empty
       $param = $element;
       if ($append eq "") {$param = "?".$param;} #Add ? to beginning
       else {$param = "&".$param;}               #Add & otherwise
       $in{$element} =~ s/([^a-zA-Z0-9_\-.])/uc sprintf("%%%02x",ord($1))/eg;  #URL Encode the value
       $append=$append.$param."=".$in{$element}; #Enter parameter
    }

    #### Send to new location

    #&print_error("SQL command","$sql\n");
    print "Status: 302 \n";
    print "Location: $in{'redirect'}$append\n\n";


    ################ END OF MAIN ################
    sub print_error
    {
       print &PrintHeader;
       print &HtmlTop("$_[0]");
       print "<pre>\n";
       print $_[1];
       print "</pre>\n";
       print &HtmlBot;
       exit;
    }

    #### Construct and Execute the SQL Insert command ####
    sub insert_sql
    {
       my ($table,$my_insert,$my_date_time,$my_dsn,$my_pwd,$my_uid) = @_;
       my ($fields,$values) = "";
       foreach $key (keys %$my_insert)
       {
          $value="'".$$my_insert{$key}."'";
          if ($fields ne "") {$key=",".$key;}
          if ($values ne "") {$value=",".$value;}
          $fields=$fields.$key;
          $values=$values.$value;
       }
       $sql="INSERT INTO ".$table." (".$fields.",created) VALUES (".$values.",'".$my_date_time."')";

       # Execute the SQL command #

       if (!($db = new Win32::ODBC("DSN=$my_dsn;UID=$my_uid;PWD=$my_pwd;"))) #Establish connection.
       {                                                                      #If error, print it
          $error = "<strong>Error connecting to $dsn</strong>\n\n";
          $error = $error."<strong>Error:</strong> \n" . Win32::ODBC::Error() . "\n";
          return $error;
       }

       $db->Sql($sql);  #Execute the command
       if (($err = Win32::ODBC::Error()) ne "")  #If there is an error, print it
       {
          $error = "<strong>Error sending SQL command:</strong><br>\n";
          $error = $error."$sql<br><br>\n";
          $error = $error."<strong>Error:</strong> \n$err\n";
          return $error;
       }
       #return "No error: ".$sql;
    }
```

Multi-stage Registration
With SQL Insert via Perl

1. Introduction

The Multi-stage registration breaks up the registration process into two stages. In the first stage, we allow the user to select a username and password. We also require the user to enter a password. In the second stage, the name, credit card info, and address (optional) are entered.

Should the user forget to enter a field in the form, a text box near the submit button prompts the user to complete the form.

The email address from the first phase of the registration is stored in a SQL database table via a Perl script. If the user does not have a JavaScript enable browser, the Perl script will also remind the user to properly fill in the form.

This document assumes an understanding of "The Mousetrap for Programmers" document.

2. Registration #1

2.1. JavaScript

The JavaScript for this page is quite involved because of the amount of checking done. The JavaScript checks the following:

1. A username has been entered.
2. A password has been entered.
3. A password made of letters and numbers has been entered.
4. A password that is 4 - 8 characters has been entered.
5. The password verify has been answered.
6. The password is the same as the password verify.
7. An email address which has: no spaces, no commas, one @, a period that is not the last character, and a period after the @.

If there are multiple errors, the message will just show the first error in the above sequence. When that error is corrected and the user pushes the submit, the JavaScript will show the next error.

The JavaScript also has a function for blinking that calls itself recursively to keep the blink cycle going. The text stays on for 1200 ms and off for 200 ms.

The following is the JavaScript for this page:

```
<script language="JavaScript">
   parent.backTo="http://www.sexmuseum.com/cgi-bin/redirect_distribute/distribute.pl";
   parent.warpTo="http://www.sexmuseum.com/cgi-bin/redirect_distribute/distribute.pl";
   parent.backToTop=true; //blow away frame
   parent.newWin=true;
   window.defaultStatus=parent.defaultStatusMsg;
   parent.loadPage();
   var isSent = 0;
   var blinking=false;

   function blinkText(aForm,txt,aFlag,rptFlag)
   {
      this.aForm=aForm; this.txt=txt; onFlag=aFlag;repeatFlag=rptFlag;
      if (!isSent && (repeatFlag || !blinking))
      {
         if (onFlag)
         {
            aForm.reminder.value=txt;
            setTimeout('blinkText(aForm,txt,!onFlag,true)', 1200);
         }
         else
         {
            aForm.reminder.value="";
            setTimeout('blinkText(aForm,txt,!onFlag,true)', 200);
         }
      }
      blinking=true;
      return;
   }

   function checkEmail(addr)
   {
      space = addr.indexOf(" ");
      comma = addr.indexOf(",");
      firstAT = addr.indexOf("@");
      lastAT = addr.lastIndexOf("@");
      lastDOT = addr.lastIndexOf(".");
      len = addr.length;
      if (
            (lastAT == -1) || (lastDOT == -1) || (lastAT != firstAT) ||
            ((lastDOT - lastAT) < 2) || ((len - lastDOT) < 2) || (space != -1) ||
            (comma != -1)
         ) return false;
      else return true;
   }
```

```
function SubmitData( form )
   {
      if(!isSent)
      {
          if (form.PARM10.value == "")
         {
            blinkText(form,"Please choose a member name.",true,false);
         }
         else if (form.PARM11.value == "")
         {
            blinkText(form,"Please enter a password.",true,false);
         }
         else if (escape(form.PARM11.value).indexOf("%") != -1)
         {
            blinkText(form,"Password: letters & numbers only.",true,false);
         }
         else if (form.PARM11.value.length < 4)
         {
            form.PARM11.value = form.PARM12.value = ""
            blinkText(form,"Password has 4 characters min.",true,false);
            blinkText(form,"Password needs 4 characters min.",true,false);
         }
         else if (form.PARM12.value == "")
         {
            blinkText(form,"Please verify password.", true);
         }
         else if ((form.PARM11.value != form.PARM12.value))
         {
            form.PARM11.value = form.PARM12.value = ""
            blinkText(form,"Password error, please re-enter.",true,false);
         }
         else if (!checkEmail(form.PARM9.value))
         {
            blinkText(form,"Please check your email address.",true,false);
         }
         else
         {
            parent.newWin=false;
            isSent = 1;
            form.reminder.value="Thank you! One moment...";
            //window.alert("submitting");
            form.submit( );
         }
      }
   }

   function load()
   {
      document.forms[0].reminder.value="Please complete before sending.";
   }

//  Script end -->
```

2.2. Register #1 Form

The hidden fields of the form configure this form to work with a particular table in a database (the database is selected by the DSN and the DSN is selected in the Perl script).

The following is the configuration section of the form:

```
<form name="qForm" method=get action="/cgi-bin/register1.pl">
 <input type=hidden name="SID" value="<#SID.#>">
 <input type=hidden name="feeder" value="<#.feeder#>">
 <input type=hidden name="bid" value="<#.bid#>">
 <input type=hidden name="js" value="<#SID.js#>">
 <input type=hidden name="referer" value="<#SID.referer#>">
 <input type=hidden name="table_name" value="register1">
 <input type=hidden name="redirect" value="http://www.sexmuseum.com/mmuse_register2.sml">
 <input type=hidden name="redirect_append"
    value="bid,PARM9,PARM10,PARM11,PARM12,referer,js,feeder">
 <input type=hidden name="stored" value="PARM9,send_email,feeder">
```

The first line issues a get to "register1.pl" when a submission occurs. The "SID", "feeder", "bid", "SID.js", and "SID.referer" fields pass these SML variables to the second registration page.

The "table_name" parameter is the name of the table in SQL_Server that will receive the data. The "redirect" is the next page to load after the Perl script runs.

The "redirect_append" parameter indicates which parameters should get appended to the "redirect" URL. Note that each parameter is separated by a comma.

The "stored" indicates which parameters need are to be stored in the database table. Note that the form "name" names need to be the same as the field names in the SQL database table.

2.3. Perl Script

The Perl script for this page verifies the same items as the JavaScript because the user may not have a JavaScript enabled browser. The Perl script also inserts the email address into an SQL database via ODBC.

2.3.1. Description

The Perl is fairly well commented so it requires little explanation here. The ODBC module must be installed on the Perl of every server that uses this Perl script. We tested this with Perl 5.001.

If there are any errors storing into a table, the Perl script displays a diagnostic error page.

2.3.2. Installing the ODBC Module

The "perl\lib\win32" directory needs to have the "odbc.pm" file installed. Also the "perl\lib\auto\win32\odbc" directory needs to be created and in that directory, the "odbc.pll" needs to be placed. These extensions were developed by Dave Roth and further info can be found at the following URLS:

```
http://multiweb.lib.calpoly.edu/odbc/
http://www.roth.net/odbc/odbcfaq.htm
http://citel2.tp.ac.sg/stfindex/staff/kinchew/odbc/odbcfaq.htm
http://www.iftech.com/oltc/webdev/webdev4.stm
http://www.ptyx.com/howto/index1.html
```

The above URL's are given for reference only... it is not necessary to read the above documentation to make the program operate. The ODBC Perl files can be downloaded at:

ftp://ftp.epix.net/pub/languages/perl/authors/Dave_Roth/Win32odbc_v970208.zip

The above file has further documentation on the ODBC modules. Currently these files are installed on the KAUAI computer.

2.3.3. The MS SQL Server Table

Before the Perl can send data to SQL Server, a table with the name "register1" needs to be setup like the following (a development table exists on PALERMO in the "[redacted]" database):

Table: register1 (dbo)

Key	Identity	Column Name	Datatype	Size	Nulls	Default
🔑		id	int	4		
		created	datetime	8	✓	
		feeder	varchar	48	✓	
		PARM9	varchar	48	✓	
		send_email	varchar	6	✓	
		[redacted]				

2.3.4. The Script

The $dsn, $uid, and $pwd variables in the script need to configured after an SQL Server database, and DSN is setup

```
#############################################################################
# REGISTER1 ver 1.1   Copyright 1998 SuperBusiness Net                      #
# Written by Wayne W. Weber January 26, 1998  Last Modified: January 26,1998 #
#                                                                           #
# Checks for valid Member Name, Password, and Email and then Inserts these  #
# Values into a Database via ODBC.                                          #
#                                                                           #
#############################################################################
$dsn="PALERMO";
$uid="sa";
$pwd="xxxxxx";
require "cgi-lib.pl";
use Win32::ODBC;

################ START OF MAIN ################

#### Parse arguments
&ReadParse(*in);

#### Flush StdOut
$| = 1;

#### Get arrays of fields that aren't required or stored ####
@stored=split(/ *, */,$in{"stored"});                    #Store these params in DB
@redirect_append=split(/ *, */,$in{"redirect_append"}); #Append params to redirect URL

#### Check for errors in entry ####
$error_found="";
if (($error_found = &check_error(*in)) ne "")
{
   &print_error("Oops... some errors were found",$error_found,);
}

#### Construct the date/time value ####
($sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdat) = localtime(time);
++$mon; $date_time = "$mon-$mday-$year $hour:$min:$sec";

#### Find fields to be stored ####
foreach $store (@stored)
{
   if ($in{$store} ne "")
   {
      $insert{$store}=$in{$store};
   }
}
```

```
#### Execute the SQL Insert command subroutine ####
if ($error_found eq "")    #Don't insert a record if form not filled out properly
{
   $error_found="";
   if (($error_found =
     &insert_sql($in{'table_name'},\%insert,$date_time,$dsn,$pwd,$uid)) ne "")
   {
      &print_error("SQL Error",$error_found);
   }
}

#### Append parameters indicated in @redirect_append ####
$append="";
foreach $element (@redirect_append)
{
   if ($in{$element} eq "") {next;}          #Don't append if empty
   $param = $element;
   if ($append eq "") {$param = "?".$param;} #Add ? to beginning
   else {$param = "&".$param;}               #Add & otherwise
   $in{$element} =~ s/([^a-zA-Z0-9_\-.])/uc sprintf("%%%02x",ord($1))/eg; #URL Encode
   $append=$append.$param."=".$in{$element}; #Enter parameter
}

#### Send to new location ####
print "Status: 302 \n";
print "Location: $in{'redirect'}$append\n\n";

################ END OF MAIN ################
################ START OF SUBROUTINES ################
sub print_error
{
   print &PrintHeader;
   print "<html><head><title>$_[0]</title></head>\n<body
BGCOLOR=#ffffd0><h2>$_[0]</h2>\n";
   print $_[1];
   print &HtmlBot;
   exit;
}
```

```
#### Check for errors in entry ####
sub check_error
{
   local(*in) = @_;

   $error_found=0; $error="";
   if ($in{'PARM10'} eq "")
   {
      $error = $error."<li>"."Please enter a member name."."\n</li>";
   }
   elsif ($in{'PARM10'} !~ /[A-Za-z].*/)
   {
      $error = $error."<li>"."Your member name needs to start with a letter."."\n</li>";
   }
   elsif ($in{'PARM10'} =~ /\W+/)       #Does the name contain any non-alpha chars
   {
      $error =
         $error."<li>"."Your member name can have only letters and numbers."."\n</li>";
   }

   if ($in{'PARM11'} eq "")
   {
      $error = $error."<li>"."Please enter a PASSWORD."."\n</li>";
   }
   elsif ($in{'PARM11'} !~ /\w{4,8}/)  #Is the password at least 4 chars
   {
      $error =
        $error."<li>"."Your password needs to be 4 characters or longer."."\n</li>";
   }
   elsif ($in{'PARM11'} =~ /\W+/)       #Does the password contain any non-alpha chars
   {
      $error =
        $error."<li>"."The password can have only letters and numbers."."\n</li>";
   }

   if ($in{'PARM12'} eq "")
   {
      $error = $error."<li>"."Please enter password VERIFY."."\n</li>";
   }
   elsif (($in{'PARM11'} ne "") &&  ($in{'PARM11'} ne $in{'PARM12'}))
   {
      $error =
        $error."<li>"."Your PASSWORD does not match the VERIFY password."."\n</li>";
   }

   if ($in{'PARM9'} eq "")
   {
      $error = $error."<li>"."Please enter your EMAIL address."."\n</li>";
   }
   elsif ($in{'PARM9'} !~ /.+\@.+\..+/)
   {
      $error =
       $error."<li>"."There appears to be a problem with your EMAIL address."."\n</li>";
   }

   if ($error ne "")
   {
      $error=$error."</ul>\n\n<strong>Please press your browser's BACK BUTTON and
correct the error.</strong>";
   }
   return $error;
}
```

```
#### Construct and Execute the SQL Insert command ####
sub insert_sql
{
   my ($table,$my_insert,$my_date_time,$my_dsn,$my_pwd,$my_uid) = @_;
   my ($fields,$values) = "";
   foreach $key (keys %$my_insert)
   {
      $value="'".$$my_insert{$key}."'";
      if ($fields ne "") {$key=",".$key;}
      if ($values ne "") {$value=",".$value;}
      $fields=$fields.$key;
      $values=$values.$value;
   }
   $sql="INSERT INTO ".$table." (".$fields.",created) VALUES
(".$values.",'".$my_date_time."')";

   # Execute the SQL command #

   if (!($db = new Win32::ODBC("DSN=$my_dsn;UID=$my_uid;PWD=$my_pwd;")))#Make connection
   {                                                                     #if error, print it
      $error = "<strong>Error connecting to $dsn</strong>\n\n";
      $error = $error."<strong>Error:</strong> \n" . Win32::ODBC::Error() . "\n";
      return $error;
   }

   $db->Sql($sql);  #Execute the command
   if (($err = Win32::ODBC::Error()) ne "")  #If there is an error, print it
   {
      $error = "<strong>Error sending SQL command:</strong><br>\n";
      $error = $error."$sql<br><br>\n";
      $error = $error."<strong>Error:</strong> \n$err\n";
      return $error;
   }
   #return "No error: ".$sql;
}
```

3. Registration #2

This page receives parameters from the first registration page that were passed on by the Perl script. SML at the top of this page is used to assign local parameters to SID parameters so that a standard form could be used (which expects "SID.referer" for example).

We developed two versions of this page, one which does not ask for address information and one which does. The file "register2.sml" does not ask for address information while "register2a.sml" does.

3.1. SML

The following SML should be at the top of the registration page:

```
<$assign val:SID.mm="apps.xpics.com">
<$if cond:.free!=CHECKED cond:.mall!=CHECKED val:.m90=CHECKED><$endif>
<$if cond:.bid val:.acb=.bid><$endif>
<$if cond:.referer val:SID.referer=.referer><$endif>
<$if cond:.js val:SID.js=.js><$endif>
```

3.2. JavaScript

Like the first registration page, this page also verifies the user entry before a submission is allowed. A function also checks a credit card for validity. The following JavaScript is used:

```
<script language="JavaScript">
<!-- Script start
var isSent = 0;
var blinking=false;

function blinkText(aForm,txt,aFlag,rptFlag)
{
   this.aForm=aForm; this.txt=txt; onFlag=aFlag;repeatFlag=rptFlag;
   if (!isSent && (repeatFlag || !blinking))
   {
      if (onFlag)
      {
         aForm.reminder.value=txt;
         setTimeout('blinkText(aForm,txt,!onFlag,true)', 1200);
      }
      else
      {
         aForm.reminder.value="";
         setTimeout('blinkText(aForm,txt,!onFlag,true)', 200);
      }
   }
   blinking=true;
   return;
}
```

```
function isCreditCard(cc)
{
   // Encoding only works on cards with less than 19 digits
   l=cc.length;st="";
   for (i = 0; i < cc.length; i++) //remove spaces
   {
      if (cc.substring(i,i+1) == " ") continue;
      st = st + cc.substring(i,i+1);
   }

   if (st.length > 19) return (false);
   sum = 0; mul = 1; l = st.length;
   for (i = 0; i < l; i++)
   {
      digit = st.substring(l-i-1,l-i);
      tproduct = parseInt(digit ,10)*mul;
      if (tproduct >= 10) sum += (tproduct % 10) + 1;
      else sum += tproduct;
      if (mul == 1) mul++;
      else mul--;
   }

   if ((sum % 10) == 0) return (true);
   else return (false);
}

function SubmitData( form )
{
   if(!isSent)
   {
      if (form.PARM7.value == "")
      {
         blinkText(form,"Please enter your first name.",true, false);
      }
      else if (form.PARM8.value == "")
      {
         blinkText(form,"Please enter your last name.",true,false);
      }
      else if (form.PARM14.value == "")
      {
         blinkText(form,"Please enter your credit card.",true,false);
      }
      else if (form.PARM15.selectedIndex == "0")
      {
        blinkText(form,"Please select expiration month.",true,false);
      }
      else if (form.PARM16.selectedIndex == "0")
      {
         blinkText(form,"Please select expiration year.",true,false);
      }
      else if (form.PARM17.selectedIndex != "1")
      {
         blinkText(form,"Please answer: Do you accept...",true,false);
      }
      else if (!isCreditCard(form.PARM14.value))
      {
         blinkText(form,"One moment please...", true);
         setTimeout('blinkText(document.forms[0],"Please enter a valid
credit card.",true,false);',5000);
      }
      else
```

```
        {
            parent.newWin=false;
            isSent = 1;
            form.reminder.value="Thank you! One moment...";
            //window.alert("submitting: ");
            form.submit( );
        }
    }
}

function load()
{
    document.forms[0].reminder.value="Please complete before entering.";
}
// Script end -->
</script>
```

```
/* Microsoft SQL Server - Scripting     */
/* Server: PALERMO          */
/* Database: xpl_maindb_f          */
   Creation Date 3/31/98 3:13:54 PM        */

use PreRegister
GO

/****** Object:  Table dbo.PreUsers    Script Date: 3/31/98 3:13:55 PM ******/
if exists (select * from sysobjects where id = object_id('dbo.PreUsers') and sysstat & 0xf = 3)
 drop table dbo.PreUsers
GO

/****** Object:  Table dbo.PreUsers    Script Date: 3/31/98 3:13:55 PM ******/
CREATE TABLE dbo.PreUsers (
 Address1 varchar (40) NULL ,
 Address2 varchar (40) NULL ,
 City varchar (40) NULL ,
 Country varchar (40) NULL ,
 Phone varchar (20) NULL ,
 State varchar (40) NULL ,
 USER_AGANT varchar (64) NULL ,
 ZIP varchar (20) NULL ,
 banner_id varchar (32) NULL ,
 created datetime NOT NULL ,
 email varchar (64) NULL ,
 email_ok smallint NULL ,
 feder varchar (40) NULL ,
 st_name varchar (40) NULL ,
 ip varchar (32) NULL ,
 last_modified datetime NOT NULL ,
 last_name varchar (40) NULL ,
 pri_id varchar (32) NOT NULL ,
 referer varchar (128) NULL
)
GO

 CREATE  INDEX ID_PreUsers_email ON dbo.PreUsers(email)
GO

 CREATE  INDEX ID_PreUsers_fn ON dbo.PreUsers(first_name)
GO

 CREATE  INDEX ID_PreUsers_ln ON dbo.PreUsers(last_name)
GO


 CREATE  INDEX ID_PreUsers_zip ON dbo.PreUsers(ZIP)
GO

 CREATE  CLUSTERED  INDEX IDUSER_Precreated ON dbo.PreUsers(created)
GO
```

Thumbnail Indexer

Thumbnail Indexer

1. Introduction

This program, written in Java 1.0, selects images out of a storage area and moves them to a viewing area. It should be run on a scheduled basis to rotate through the images in the storage area (this program is not an applet but a stand-alone application run from DOS).

The program's ".class" files can be run with Sun's JRE or it could be compiled to create an EXE file using SuperCede or Symantic's Java compiler.

The developer of this program (Wayne Weber) left a running sample of this program and sample image directories on a CD and on the computer KAUAI and in the server GONDOLIER on the E_drive in a directory called "Waynes_archive_SAVE"

2. Example

The example included here is for images for three fetishes: substantial, thin, and mature. The images in the example are random and not really substantial, thin, or mature. They images can be located in the "d:\http\sd\membership\members\fetish" directory.

The file called "indexer.ini" has information for configuring the program. The file "indexer.dat" has data for the last copied files.

When the "indexer.exe" program in the "d:\java\Fetish_c\" directory is run, the next series of 50 images will be copied out of:

```
D:\http\sd\membership\members\fetish\Mature\payoff\reserve\
```

to:

```
D:\http\sd\membership\members\fetish\Mature\payoff\
```

and from:

```
D:\http\sd\membership\members\fetish\Mature\closeup\reserve\
```

to:

```
D:\http\sd\membership\members\fetish\Mature\closeup\
```

This will also happen in parallel directories for the "substantial", and "thin" categories.

Note: Instead of running the "indexer.exe" program, we also have a batch file that runs Sun's JRE to execute the ".class" files. That batch file is called "jre.bat" and is located in the "d:\java\Fetish_c\Indexer\ directory. That batch file looks like the following:

```
\java\jre1.1\bin\jre -cp \java\Fetish_c\Indexer Indexer
```

After the files have been copied, on Kauai, they can be viewed at:

```
http://kauai.sbusiness.com/sd/membership/members/fetish/index.sml
```

3. SML files

Not only does the "indexer" program copy files, it also creates the thumbnail pages for each category. When the thumbnail is clicked, the link passes parameters to a generic SML page, "Showpayoff.sml", that loads the proper full sized image. Also passed to the "Showpayoff.sml" page is information indicating if a closeup exists. If a closeup does exist, a link appears on the "Showpayoff.sml" page to the "Showcloseup.sml" page.

The following files contain the thumbnails and links to the "Showpayoff.sml" page (and are generated by the Java program):

```
Mature.sml
Fetish.sml
Substantial.sml
```

The following files are built by the developer (we have samples):

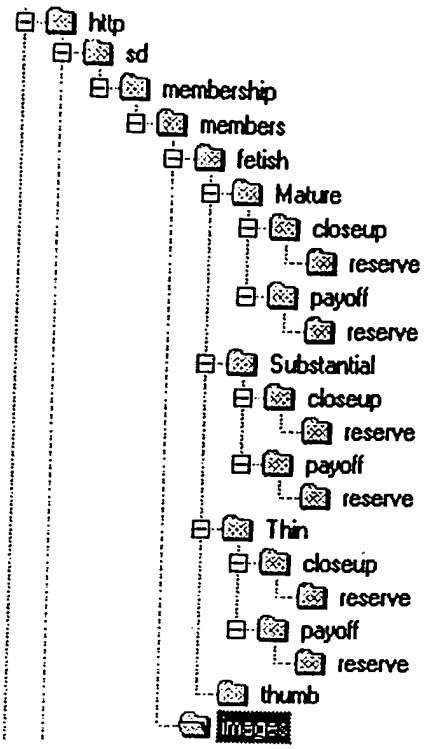
```
index.sml (or whatever name you want to give the first page)
Showpayoff.sml
Showcloseup.sml
```

The "index.sml" has links to the files generated by the "indexer" Java program... in this case, the Mature.sml, Fetish.sml, and Substantial.sml files.

The number of thumbnails and category names are set in the "indexer.ini" file.

4. Directory Structure

A directory structure must be followed for the program to work properly. The following is the structure for our example:



An entry in the "indexer.ini" file points to the "D:/http/sd/membership/members/fetish" directory (imageGroupsRootDir). The main directories under the "fetish" directory (Mature, Substantial, and Thin) are pointed to by the "imageGroupName" entries in the "indexer.ini" directories. The Mature, Substantial, Thin, directories and their subdirectories (payoff & reserve and closeup & reserve) must be manually created.

The image pool is placed in the reserve directories. The "indexer" program copies the next 50 (or whatever number is set as "thumbsPerIndex" in the "indexer.ini" file) to the directory above the reserve directory. This is done for the payoff and closeup (if there is a closeup available) directories.

All thumbnails for all images are in the "thumb" directory.

The filenames must be the SAME for all three image sizes. For example, a file in the "Mature/payoff/reserve" directory called "mat1211.jpg" would also be called "mat1211.jpg" in the Mature/closeup/reserve" directory. In the "thumb" directory, it would also be called "mat1211.jpg".

Note that the image files MUST end in ".jpg".

5. Operation

The "indexer" program sequences through each of the image group directories. In our example, it might first go to the "Mature" directory and then sequence through the following:

1. Get the alphabetic listing of the files in its "Mature/payoff/reserve" directory.
2. Examine the "indexer.dat" file to see what was the last file copied from the "payoff/reserve" directory.
3. Delete all files in the "Mature/payoff" directory.
4. Copy the next 50 files from the next "Mature/payoff/reserve" dirctory to the "Mature/payoff" directory.
5. Delete all files in the "Mature/closeup" directory.
6. Look in the "Mature/closeup/reserve" directory and see which of the 50 files found in the Mature/payoff/reserve" are also present in the "Mature/closeup/reserve" directory.
7. Copy the files found in step 6 to the "Mature/closeup" directory.
8. Look in the "thumbnail" directory and see which thumbnails are missing.
9. Generate the "Mature.sml" page. If thumbnail missing, use generic thumbnail. If "closeup" file present, add the parameter "cl=t" in the link to the "Showpayoff.sml" page.

The above steps are repeated for each of the image group directories. If all images have been cycled through in a "reserve" directory, "indexer" starts again with the first image.

6. Indexer.ini

This file customizes "indexer". A sample file is shown below:

```
#The number of Thumbnails on an index page.
thumbsPerIndex=50

#This is the path where all the image directories are under.
#Can use back or forward slashes to separate directories
imageGroupsRootDir=D:/http/sd/membership/members/fetish

#These are the various categories of image groups.
#Do not skip a number in sequence.
#Capitalize the first letter... it is used in a title as well.
imageGroupName[0]=Mature
imageGroupName[1]=Substantial
imageGroupName[2]=Thin
```

The comments in this file are self explanatory. The "imageGroupName" is used to locate a directory and is also used in a heading on thumbnails page (so it probably should start with a capitol letter).

7. Indexer.dat

This file must initially be created manually by the developer. As the comments in the file show, it can initially be created with just the comments. The first time "indexer" is run, the "...lastFileCopied" for each image group will be added to the file.

```
#This file must exist and it is updated by the indexer.
#It can be empty except leave these comment lines in.
MatureLastFileCopied=cb006o.jpg
SubstantialLastFileCopied=cb014b.jpg
ThinLastFileCopied=mh100b.jpg
```

This file is used to keep track of the last file copied.

8. Indexer's Files

The following ".class" files are need to run indexer using Sun's JRE (the source or ".java" files are included on the CD or GONDOLIER as mentioned in the Introduction):

```
Directory.class
DirectoryManager.class
Indexer.class
JPEGFileFilter.class
ParsedFile.class
ThumbNailsPage.class
```

As described earlier, the following files are needed by indexer:

```
indexer.dat
indexer.ini
```

Temp = 0; iTemp <= sNumOfCols; iTemp++){ if (szColumn[iTemp]) delete [] szColumn[iTemp]; } delete [] szColumn; } } void CResults::Clean(){ int iTemp; for(iTemp = 1; iTemp <= sNumOfCols; iTemp++){ memset(szColumn[iTemp], 0x00, (dSize[iTemp] * dRowSetSize)); } } DWORD CResults::RowSetSize(){ return dRowSetSize; }

Win32::ODBC - Object

Contents

Part I

Part II

Creating an ODBC Object

Your script will need to have the following line:

```
use Win32::ODBC;
```

Then you will need to create a data connection to your DSN:

```
$Data = new Win32::ODBC("MyDSN");
```

You shoud check to see if *$Data* is indeed defined otherwise there has been an error. You can now send SQL queries and retrieve info to your heart's content! See the description of functions below and also test.pl to see how it all works.

Make sure that you close your connection when you are finished:

```
$Data->Close();
```

Object Methods

General Note

All methods assume that you have the line:

```
use Win32::ODBC;
```

somewhere before the method calls, and that you have an ODBC object called $db which was created using some call similar to:

```
$db = new Win32::ODBC("MyDSN");
```

See new for more information.

Also, in an effort to keep the examples short, no error checking is done on return values for any calls other than the one being exemplified. You should *always* check for error conditions in production code.

WARNING: The example code has not yet been tested. This will be fixed ASAP, but be forwarned!

Methods

Catalog *qualifier, owner, name, type*

Retrieves the catalog from the current ODBC object. Returns a four-element array (Qualifier, Owner, Name, Type). **Note:**All fieldnames are uppercase!

```
Example:
($qualifier, $owner, $name, $type) = $db->Catalog("", "", "%", "'TABLE'");
```

Connection

Returns the object's ODBC connection number.

```
Example:
$cnum = $db->Connection;
```

Close

Closes the ODBC connection for this object. It always returns **undef**.

```
Example:
$db->Close();
```

Data
Data *list*

Retrieve data from previous fetch for a list of field names. In a scalar context it returns all of the field values concatenated together. In an array context, it returns an array of the values, in the order in which they were specified. If no field names are given, all fields are returned in an unspecified order.

```
Example:
$db->Sql("SELECT f1, f2, f3 FROM foo");
$db->FetchRow();
($f1, $f2) = $db->Data("f1", "f2");

or

$db->Sql("SELECT * FROM foo");
$db->FetchRow();
@values = $db->Data;
```

See also: DataHash

DataHash
DataHash *list*

Retrieve data from previous fetch for a list of field names. Returns a hash where the field name is the key. If no field names are given, all fields are returned.

```
Example:
$db->Sql("SELECT f1, f2, f3 FROM foo");
$db->FetchRow();
%hash = $db->DataHash("f1", "f2");
print $hash{f1};

or

$db->Sql("SELECT * FROM foo");
$db->FetchRow();
%hash = $db->DataHash;
foreach $key (sort(keys %hash)) {
        print $key, '=', $hash{$key}, "\n";
}
```

See also: Data

DataSources

Returns an associative array of Data Sources and ODBC remarks in the form of:

```
$ArrayName{'DSN'} = Remark
```

where DSN is the Data Source Name and Remark is, well, the remark.

```
Example:
%rem = $db->DataSources;
print LOG qq(Current DSN's Remark: "), %rem{$db->GetDSN}, qq("\n);
```

Drivers

Returns an associative array of Drivers and their attributes in the form of:

```
$ArrayName{'DRIVER'} = Attrib1;Attrib2;Attrib3;...
```

where DRIVER is the ODBC Driver Name and AttribX are the driver-defined attributes.

```
Example:
%attrib = $db->Drivers;
print LOG qq($driver: $attrib{$driver}\n) foreach $driver (keys %attrib);
```

DumpError

Dump to the screen details about the last error condition. This includes error number, error text and the ODBC connection number that caused the error (if there is one). This is used primarily for debugging.

```
Example:
$db = new Win32::ODBC("My DSN");
if (undef $db){
    Win32::ODBC::DumpError();
}
if ($db->Sql("Select * FROM foo")){
    $db->DumpError;
}
```

DumpData

Dump to the screen all field names and the data in all rows of the current dataset. This is used primarily for debugging.

```
Example:
$db->Sql("Select * FROM foo");
$db->DumpData;
```

Error

Returns the last recorded error in the form of an array or string (depending upon the context) containing the error number, error text and the ODBC connection that caused the error (if there is one).

```
Example:
die $db->Error(), qq(\n);

($ErrNum, $ErrText, $ErrConn) = $db->Error();
```

FetchRow

Fetches the next row of data from the previous specified SQL statement. You would then call Data or DataHash to actually retrieve the individual elements of data. Returns **undef** if there's an error, **TRUE** otherwise.

```
Example:
$db->Sql("SELECT * FROM foo");
$db->FetchRow() || die qq(Fetch error: ), $db->Error(), qq(\n);
$f1 = $db->Data("f1");
```

See also: Sql, Data, DataHash

FieldNames

Returns a list of field names extracted from the current dataset. This is used mostly for testing/debugging. FieldNames returns the data in an array, with no guarantee of the order of the names.

```
Example:
$db->Sql("SELECT * FROM foo");
$db->FetchRow();
foreach $fd ($db->FieldNames()) print qq($fd: "), $db->Data($fd), qq("\n);
```

GetConnections

Returns an array of connection numbers for all objects.

```
Example:
@cnums = $db->GetConnections;
```

GetDSN
GetDSN *conn*

Returns the DSN (Data Source Name) or the ODBCDriverConnect string for the connection *conn*, or the current connection if not specified.

```
Example:
print LOG qq(Current connection: "), $db->GetDSN, qq("\n);
```

GetMaxBufSize

Returns the current maximum single field data size, in bytes.

```
Example:
$max = $db->GetMaxBufSize;
$db->SetMaxBufSize($needed) if ($max < $needed);
```

See also: SetMaxBufSize

GetStmtCloseType

Returns the current ODBC close type setting. This is used mainly for debugging. Type will be one of: SQL_CLOSE, SQL_DROP, SQL_UNBIND, or SQL_RESET_PARAMS. See SetStmtCloseType for more info on what each of the types mean, and how they are used.

```
Example:
$oldct = $db->GetStmtCloseType;
$db->SetStmtCloseType(SQL_DROP);
...
$db->SetStmtCloseType($oldct);
```

See also: SetStmtCloseType

MoreResults

Sees if more result sets are present and initializes for fetching rows from next result set. You would then call FetchRow to actually fetch the next row of the next result set. Returns **undef** if there's an error, **TRUE** otherwise.

```
Example:
$db->Sql("SELECT * FROM foo\n  SELECT * FROM bar");
$db->FetchRow() || die qq(Fetch error: ), $db->Error(), qq(\n);
$f1 = $db->Data("f1");
$db->MoreResults() || die qq(Error checking for more result sets: ), $db->Error
```

```
$db->FetchRow() || die qq(Fetch error: ), $db->Error(), qq(\n);
$f1 = $db->Data("f1");
```

See also: Sql, Data

new Win32::ODBC(DSN)
new Win32::ODBC(ODBCDriverConnect)

Creates a new ODBC object, given a DSN (Data Source Name) or a properly formatted ODBCDriverConnect string. Returns the created ODBC object or **undef** if there is an error.

```
Example:
$DSN = "MyDSN";
$db = new Win32::ODBC($DSN);
die qq(Cannot open new ODBC\n) if ! $db;

or

$db = new Win32::ODBC("dsn=FOO;UID=BAR;PWD=FUBAR");
die qq(Cannot open new ODBC\n) if ! $db;
```

RowCount

Returns the number of rows that were affected by the previous SQL command. **Note:** This does not work on all ODBC connections.

```
Example:
$db->Sql("SELECT * FROM foo");
print DBG q(# of records: ), $db->RowCount(), qq(\n);
```

Run *stmt*

Submit the SQL statement *stmt* and print data about it. This is used only in debugging.

```
Example:
$db->Run("SELECT * FROM foo");
```

See also: Sql

<!C-

Type">SetStmtCloseType *type*

Sets the current ODBC close type setting used by the ODBC Manager. This is used mainly for debugging. Normally, when you open a statement handle and perform a query (or whatever) the results are associated with the statement. You need to free the statement in order to execute another query. When you do this, usually the dataset (from the query) is cached. This caching action may be good for speed but could cause some memory problems if your dataset is huge. See the ODBC API call **SQLFreeStmt(hstmt, option)** for more details. (All of this is handled automatically by the Win32::ODBC package).

Type will be one of:

- SQL_CLOSE - just close the statement (use caching)
- SQL_DROP - close and drop all results (do not use caching)
- SQL_UNBIND - close and remove bindings to columns (odbc.pll does not bind vars to columns)
- SQL_RESET_PARAMS - close and reset all of the bound parameters (such as type casting for columns; see **SQLFreeStmt()**)

```
Example:
$oldct = $db->GetStmtCloseType;
$db->SetStmtCloseType(SQL_DROP);
...
$db->SetStmtCloseType($oldct);
```

See also: GetStmtCloseType

ShutDown

Closes the ODBC connection and print data about it. This is used only in debugging.

```
Example:
$db->Shutdown;
```

See also: Close

Sql *stmt*

Executes the SQL command *stmt*. Returns **undef** on success, SQL error code on failure.

```
Example:
$stmt = "SELECT * FROM foo";
$rc = $db->Sql($stmt);
die qq(SQL failed "$stmt": ), $db->Error(), qq(\n) if $rc;
```

See also: Error

TableList

TableList *qualifier, owner, name, type*

Retrieves the list of table names from the current ODBC object using Catalog. If not specified, *qualifier* and *owner* default to "", *name* defaults to "%", and *table* defaults to "'TABLE'". TableList returns an array of table names. **Note:**All fieldnames are uppercase!

```
Example:
@tables = $db->TableList;
```

See also: Catalog

Examples

This page maintined by Joe Casadonte. Please let me if something is wrong or does not make sense. Send these or other comments to: joc@netaxs.com.

Win32::ODBC - Object / 22 Jul 1996 / joc@netaxs.com

Win32::ODBC

The Win32::ODBC FAQ is now available! (*be patient, it is still under construction*)

Contents

Part I

Part II

Module Description

Business Stuff

ODBC extension for Win32 PERL
Latest version: Version 961011
by Dave Roth (rothd@roth.net)
Courtesy of Roth Consulting.

Based on code by Dan DeMaggio (dmag@umich.edu) -- Thanks Dan!

Copyright

Disclaimer

I do not guarantee ANYTHING with this package. If you use it you are doing so AT YOUR OWN RISK! I may or may not support this depending on my time schedule and I am neither an ODBC nor SQL guru so do not ask me questions regarding them!

GNU General Public License

GNU Library General Public License - GNU Project

Larry Wall's "Artistic License"

Artistic License

Version History

ODBC.PM

Date	Changes
96.03.27	Initial Release (rothd)
96.04.08	Changed Data() to accept an array of field names, returning an array or scalar (joc)
	Various bug fixes (joc)
96.04.10	Fixed the RowCount() to default to the current connection (droth)
96.04.15	Changed version numbers to a date format (rothd)
96.05.07	Fixed Data(): If returning an array, the array consisted of the requested fieldnames + the values. Now only returns the values. Thanks to Dan Westerlund <westerlund@dkraft.se>! (rothd)
96.07.22	Added MoreResults(): Checks to see if there are any more records that have not been retrieved. (dunfordshore)
96.07.22	Changed Error(): Errors are tracked differently now. Any error will record the error number ($ErrNum) and error text ($ErrText) in the object's name space and can be retrieved by calling $Database->Error(). Error() will also return the ODBC connection number that caused the error. If the context of the call is not an array then a string with this information will be returned. The Win32::ODBC name space will always have the last error info made, even if it is an error returned by the new method. You can call Win32::ODBC::Error() to get that. (rothd)

ODBC.PLL

Date	Changes
96.04.10	Fixed a "memory bug": we were using SQL_CLOSE when closing an ODBC statement. This kept the cursor alive in memory so it can be cached in the event the same SQL statement is issued. We are now using SQL_DROP. This may lessen speed if the same SQL statements are issued again and again per connection. (rothd)
	Cleaned up some code. (rothd)
96.04.12	Added GetStmtCloseType() and SetStmtCloseType() functions. (rothd)
	Added some constants. (rothd)
96.04.13	** We now are trying to include a version for builds up to and including 105 and another for builds 106 and greater (for now).
96.04.15	Fix Bug in ODBCFetchRow(): when retrieving a field with a NULL value the value from the previous field was reported. (jmk)
	Changed version numbers to a date format.
96.04.22	Fix the SDWORD wrap-around bug in ODBCFetchRow(): when a column of size 2147483647 adding 1 (for a NULL byte in szBuf) yields -2147483648 (not easy to "net UCHAR (-2147483648)"!!) (jmk)
96.04.22	Inspired by Jutta, I have increased the max limit for SetMaxBufSize() to 2,147,483,647 bytes. (rothd)
96.05.03	Set the lowest allocated char array to be 20 bytes in ODBCFetchRow(). Evidentally sometimes the ODBC manager will report too few chars that are needed to represent an autonumber field. (rothd)
96.05.08	Convert all results from ODBCTableList() to uppercase since different ODBC drivers impliment this differently (some uppercase some lower). Thanks again to Jutta M. Klebe. (rothd) *** This patch is open for suggestions!!! ***
96.07.22	Added ODBCMoreResults(): Checks to see if there are any more records that have not been retrieved. (dunfordshore)

Author List

Abbreviation	Name	EMail
rothd	Dave Roth	rothd@roth.net
joc	Joe Casadonte	joc@netaxs.com
jmk	Jutta M. Klebe	jmk@exc.bybyte.de
dunfordshore	Brian Dunfordshore	brian_dunfordshore@bridge.com

---

Features/Limits

Features

- The number of ODBC connections is limited only by memory and ODBC itself.
- The working limit to the size of a field is 10,240 bytes. This can be raised (if needed) to a maximum of 2,147,483,647 bytes (memory permitting).
- Connections can be opened using a DSN or connection string.
- Connections can be opened and closed in any order.
- Catalog (and TableList) functions are supported.
- Add, modify and remove DNS's.
- Transactions are supported (rollback and commit).
- GetInfo() is supported.
- Get/SetConnectOptions() is supported.
- Get/SetStmtOptions() is supported.
- The entire SQL_xxx constant set is supported.

Limits

- If the account that the process runs under does not have write permission on the default directory (for the process, not the ODBC DSN) you will probably get a run-time error during an SQLConnection(). I don't think that this is a problem with the code, more like ODBC. This happens because some ODBC drivers need to write a temporary file. I noticed this using the MS Jet Engine (Access Driver).
- This has not been optimized for speed nor optimized for memory consumption. This may run into memory bloat.
- Release versions are comp

```
# Perl Routines to Manipulate CGI input
# cgi-lib@pobox.com
# $Id: cgi-lib.pl,v 2.15 1997/11/18 06:48:25 brenner Exp $
#
# Copyright (c) 1997 Steven E. Brenner
# Unpublished work.
# Permission granted to use and modify this library so long as the
# copyright above is maintained, modifications are documented, and
# credit is given for any use of the library.
#
# Thanks are due to many people for reporting bugs and suggestions

# For more information, see:
#     http://cgi-lib.stanford.edu/cgi-lib/

$cgi_lib'version = sprintf("%d.%02d", q$Revision: 2.15 $ =~ /(\d+)\.(\d+)/);


# Parameters affecting cgi-lib behavior
# User-configurable parameters affecting file upload.
$cgi_lib'maxdata    = 131072;     # maximum bytes to accept via POST - 2^17
$cgi_lib'writefiles =      0;     # directory to which to write files, or
                                  # 0 if files should not be written
$cgi_lib'filepre    = "cgi-lib"; # Prefix of file names, in directory above

# Do not change the following parameters unless you have special reasons
$cgi_lib'bufsize  =  8192;     # default buffer size when reading multipart
$cgi_lib'maxbound =   100;     # maximum boundary length to be encounterd
$cgi_lib'headerout =    0;     # indicates whether the header has been printed


# ReadParse
# Reads in GET or POST data, converts it to unescaped text, and puts
# key/value pairs in %in, using "\0" to separate multiple selections

# Returns >0 if there was input, 0 if there was no input
# undef indicates some failure.

# Now that cgi scripts can be put in the normal file space, it is useful
# to combine both the form and the script in one place.  If no parameters
# are given (i.e., ReadParse returns FALSE), then a form could be output.

# If a reference to a hash is given, then the data will be stored in that
# hash, but the data from $in and @in will become inaccessable.
# If a variable-glob (e.g., *cgi_input) is the first parameter to ReadParse,
# information is stored there, rather than in $in, @in, and %in.
# Second, third, and fourth parameters fill associative arrays analagous to
# %in with data relevant to file uploads.

# If no method is given, the script will process both command-line arguments
# of the form: name=value and any text that is in $ENV{'QUERY_STRING'}
# This is intended to aid debugging and may be changed in future releases

sub ReadParse {
  local (*in) = shift if @_;     # CGI input
  local (*incfn,                 # Client's filename (may not be provided)
         *inct,                  # Client's content-type (may not be provided)
         *insfn) = @_;           # Server's filename (for spooled files)
  local ($len, $type, $meth, $errflag, $cmdflag, $perlwarn, $got);

  # Disable warnings as this code deliberately uses local and environment
  # variables which are preset to undef (i.e., not explicitly initialized)
  $perlwarn = $^W;
```

```
    $^W = 0;

    binmode(STDIN);   # we need these for DOS-based systems
    binmode(STDOUT);  # and they shouldn't hurt anything else
    binmode(STDERR);

    # Get several useful env variables
    $type = $ENV{'CONTENT_TYPE'};
    $len  = $ENV{'CONTENT_LENGTH'};
    $meth = $ENV{'REQUEST_METHOD'};

    if ($len > $cgi_lib'maxdata) { #'
        &CgiDie("cgi-lib.pl: Request to receive too much data: $len bytes\n");
    }

    if (!defined $meth || $meth eq '' || $meth eq 'GET' ||
        $type eq 'application/x-www-form-urlencoded') {
      local ($key, $val, $i);

      # Read in text
      if (!defined $meth || $meth eq '') {
        $in = $ENV{'QUERY_STRING'};
        $cmdflag = 1;  # also use command-line options
      } elsif($meth eq 'GET' || $meth eq 'HEAD') {
        $in = $ENV{'QUERY_STRING'};
      } elsif ($meth eq 'POST') {
          if (($got = read(STDIN, $in, $len) != $len))
            {$errflag="Short Read: wanted $len, got $got\n";};
      } else {
        &CgiDie("cgi-lib.pl: Unknown request method: $meth\n");
      }

      @in = split(/[&;]/,$in);
      push(@in, @ARGV) if $cmdflag; # add command-line parameters

      foreach $i (0 .. $#in) {
        # Convert plus to space
        $in[$i] =~ s/\+/ /g;

        # Split into key and value.
        ($key, $val) = split(/=/,$in[$i],2); # splits on the first =.

        # Convert %XX from hex numbers to alphanumeric
        $key =~ s/%([A-Fa-f0-9]{2})/pack("c",hex($1))/ge;
        $val =~ s/%([A-Fa-f0-9]{2})/pack("c",hex($1))/ge;

        # Associate key and value
        $in{$key} .= "\0" if (defined($in{$key})); # \0 is the multiple separator
        $in{$key} .= $val;
      }

    } elsif ($ENV{'CONTENT_TYPE'} =~ m#^multipart/form-data#) {
      # for efficiency, compile multipart code only if needed
$errflag = !(eval <<'END_MULTIPART');

    local ($buf, $boundary, $head, @heads, $cd, $ct, $fname, $ctype, $blen);
    local ($bpos, $lpos, $left, $amt, $fn, $ser);
    local ($bufsize, $maxbound, $writefiles) =
      ($cgi_lib'bufsize, $cgi_lib'maxbound, $cgi_lib'writefiles);


    # The following lines exist solely to eliminate spurious warning messages
    $buf = '';
```

```
    ($boundary) = $type =~ /boundary="([^"]+)"/; #";    # find boundary
    ($boundary) = $type =~ /boundary=(\S+)/ unless $boundary;
    &CgiDie ("Boundary not provided: probably a bug in your server")
      unless $boundary;
    $boundary =  "--" . $boundary;
    $blen = length ($boundary);

    if ($ENV{'REQUEST_METHOD'} ne 'POST') {
      &CgiDie("Invalid request method for  multipart/form-data: $meth\n");
    }

    if ($writefiles) {
      local($me);
      stat ($writefiles);
      $writefiles = "/tmp" unless  -d _ && -w _;
      # ($me) = $0 =~ m#([^/]*)$#;
      $writefiles .= "/$cgi_lib'filepre";
    }

    # read in the data and split into parts:
    # put headers in @in and data in %in
    # General algorithm:
    #   There are two dividers: the border and the '\r\n\r\n' between
    # header and body.  Iterate between searching for these
    #   Retain a buffer of size(bufsize+maxbound); the latter part is
    # to ensure that dividers don't get lost by wrapping between two bufs
    #   Look for a divider in the current batch.  If not found, then
    # save all of bufsize, move the maxbound extra buffer to the front of
    # the buffer, and read in a new bufsize bytes.  If a divider is found,
    # save everything up to the divider.  Then empty the buffer of everything
    # up to the end of the divider.  Refill buffer to bufsize+maxbound
    #   Note slightly odd organization.  Code before BODY: really goes with
    # code following HEAD:, but is put first to 'pre-fill' buffers.  BODY:
    # is placed before HEAD: because we first need to discard any 'preface,'
    # which would be analagous to a body without a preceeding head.

   $left = $len;
  PART: # find each part of the multi-part while reading data
   while (1) {
     die $@ if $errflag;

     $amt = ($left > $bufsize+$maxbound-length($buf)
             ?  $bufsize+$maxbound-length($buf): $left);
     $errflag = (($got = read(STDIN, $buf, $amt, length($buf))) != $amt);
     die "Short Read: wanted $amt, got $got\n" if $errflag;
     $left -= $amt;

     $in{$name} .= "\0" if defined $in{$name};
     $in{$name} .= $fn if $fn;

     $name=~/([-\w]+)/;  # This allows $insfn{$name} to be untainted
     if (defined $1) {
       $insfn{$1} .= "\0" if defined $insfn{$1};
       $insfn{$1} .= $fn if $fn;
     }

   BODY:
    while (($bpos = index($buf, $boundary)) == -1) {
      if ($left == 0 && $buf eq '') {
        foreach $value (values %insfn) {
          unlink(split("\0",$value));
        }
```

```
      &CgiDie("cgi-lib.pl: reached end of input while seeking boundary " .
              "of multipart. Format of CGI input is wrong.\n");
    }
    die $@ if $errflag;
    if ($name) {  # if no $name, then it's the prologue -- discard
      if ($fn) { print FILE substr($buf, 0, $bufsize); }
      else     { $in{$name} .= substr($buf, 0, $bufsize); }
    }
    $buf = substr($buf, $bufsize);
    $amt = ($left > $bufsize ? $bufsize : $left); #$maxbound==length($buf);
    $errflag = (($got = read(STDIN, $buf, $amt, length($buf))) != $amt);
    die "Short Read: wanted $amt, got $got\n" if $errflag;
    $left -= $amt;
  }
  if (defined $name) {  # if no $name, then it's the prologue -- discard
    if ($fn) { print FILE substr($buf, 0, $bpos-2); }
    else     { $in {$name} .= substr($buf, 0, $bpos-2); } # kill last \r\n
  }
  close (FILE);
  last PART if substr($buf, $bpos + $blen, 2) eq "--";
  substr($buf, 0, $bpos+$blen+2) = '';
  $amt = ($left > $bufsize+$maxbound-length($buf)
          ? $bufsize+$maxbound-length($buf) : $left);
  $errflag = (($got = read(STDIN, $buf, $amt, length($buf))) != $amt);
  die "Short Read: wanted $amt, got $got\n" if $errflag;
  $left -= $amt;


  undef $head;  undef $fn;
HEAD:
  while (($lpos = index($buf, "\r\n\r\n")) == -1) {
    if ($left == 0  && $buf eq '') {
      foreach $value (values %insfn) {
        unlink(split("\0",$value));
      }
      &CgiDie("cgi-lib: reached end of input while seeking end of " .
              "headers. Format of CGI input is wrong.\n$buf");
    }
    die $@ if $errflag;
    $head .= substr($buf, 0, $bufsize);
    $buf = substr($buf, $bufsize);
    $amt = ($left > $bufsize ? $bufsize : $left); #$maxbound==length($buf);
    $errflag = (($got = read(STDIN, $buf, $amt, length($buf))) != $amt);
    die "Short Read: wanted $amt, got $got\n" if $errflag;
    $left -= $amt;
  }
  $head .= substr($buf, 0, $lpos+2);
  push (@in, $head);
  @heads = split("\r\n", $head);
  ($cd) = grep (/^\s*Content-Disposition:/i, @heads);
  ($ct) = grep (/^\s*Content-Type:/i, @heads);

  ($name) = $cd =~ /\bname="([^"]+)"/i; #";
  ($name) = $cd =~ /\bname=([^\s:;]+)/i unless defined $name;

  ($fname) = $cd =~ /\bfilename="([^"]*)"/i; #"; # filename can be null-str
  ($fname) = $cd =~ /\bfilename=([^\s:;]+)/i unless defined $fname;
  $incfn{$name} .= (defined $in{$name} ? "\0" : "") .
    (defined $fname ? $fname : "");

  ($ctype) = $ct =~ /^\s*Content-type:\s*"([^"]+)"/i;  #";
  ($ctype) = $ct =~ /^\s*Content-Type:\s*([^\s:;]+)/i unless defined $ctype;
  $inct{$name} .= (defined $in{$name} ? "\0" : "") . $ctype;
```

```
        if ($writefiles && defined $fname) {
          $ser++;
          $fn = $writefiles . ".$$.$ser";
          open (FILE, ">$fn") || &CgiDie("Couldn't open $fn\n");
          binmode (FILE);  # write files accurately
        }
        substr($buf, 0, $lpos+4) = '';
        undef $fname;
        undef $ctype;
      }

  1;
  END_MULTIPART
      if ($errflag) {
        local ($errmsg, $value);
        $errmsg = $@ || $errflag;
        foreach $value (values %insfn) {
          unlink(split("\0",$value));
        }
        &CgiDie($errmsg);
      } else {
        # everything's ok.
      }
    } else {
      &CgiDie("cgi-lib.pl: Unknown Content-type: $ENV{'CONTENT_TYPE'}\n");
    }

    # no-ops to avoid warnings
    $insfn = $insfn;
    $incfn = $incfn;
    $inct  = $inct;

    $^W = $perlwarn;

    return ($errflag ? undef :  scalar(@in));
  }


  # PrintHeader
  # Returns the magic line which tells WWW that we're an HTML document

  sub PrintHeader {
    return "Content-type: text/html\n\n";
  }


  # HtmlTop
  # Returns the <head> of a document and the beginning of the body
  # with the title and a body <h1> header as specified by the parameter

  sub HtmlTop
  {
    local ($title) = @_;

    return <<END_OF_TEXT;
  <html>
  <head>
  <title>$title</title>
  </head>
  <body>
  <h1>$title</h1>
  END_OF_TEXT
```

```
}

# HtmlBot
# Returns the </body>, </html> codes for the bottom of every HTML page

sub HtmlBot
{
  return "</body>\n</html>\n";
}


# SplitParam
# Splits a multi-valued parameter into a list of the constituent parameters

sub SplitParam
{
  local ($param) = @_;
  local (@params) = split ("\0", $param);
  return (wantarray ? @params : $params[0]);
}


# MethGet
# Return true if this cgi call was using the GET request, false otherwise

sub MethGet {
  return (defined $ENV{'REQUEST_METHOD'} && $ENV{'REQUEST_METHOD'} eq "GET");
}


# MethPost
# Return true if this cgi call was using the POST request, false otherwise

sub MethPost {
  return (defined $ENV{'REQUEST_METHOD'} && $ENV{'REQUEST_METHOD'} eq "POST");
}


# MyBaseUrl
# Returns the base URL to the script (i.e., no extra path or query string)
sub MyBaseUrl {
  local ($ret, $perlwarn);
  $perlwarn = $^W; $^W = 0;
  $ret = 'http://' . $ENV{'SERVER_NAME'} .
         ($ENV{'SERVER_PORT'} != 80 ? ":$ENV{'SERVER_PORT'}" : '') .
         $ENV{'SCRIPT_NAME'};
  $^W = $perlwarn;
  return $ret;
}


# MyFullUrl
# Returns the full URL to the script (i.e., with extra path or query string)
sub MyFullUrl {
  local ($ret, $perlwarn);
  $perlwarn = $^W; $^W = 0;
  $ret = 'http://' . $ENV{'SERVER_NAME'} .
         ($ENV{'SERVER_PORT'} != 80 ? ":$ENV{'SERVER_PORT'}" : '') .
         $ENV{'SCRIPT_NAME'} . $ENV{'PATH_INFO'} .
         (length ($ENV{'QUERY_STRING'}) ? "?$ENV{'QUERY_STRING'}" : '');
  $^W = $perlwarn;
  return $ret;
```

```
}

# MyURL
# Returns the base URL to the script (i.e., no extra path or query string)
# This is obsolete and will be removed in later versions
sub MyURL {
  return &MyBaseUrl;
}


# CgiError
# Prints out an error message which which containes appropriate headers,
# markup, etcetera.
# Parameters:
#  If no parameters, gives a generic error message
#  Otherwise, the first parameter will be the title and the rest will
#  be given as different paragraphs of the body

sub CgiError {
  local (@msg) = @_;
  local ($i,$name);

  if (!@msg) {
    $name = &MyFullUrl;
    @msg = ("Error: script $name encountered fatal error\n");
  };

  if (!$cgi_lib'headerout) { #')
    print &PrintHeader;
    print "<html>\n<head>\n<title>$msg[0]</title>\n</head>\n<body>\n";
  }
  print "<h1>$msg[0]</h1>\n";
  foreach $i (1 .. $#msg) {
    print "<p>$msg[$i]</p>\n";
  }

  $cgi_lib'headerout++;
}


# CgiDie
# Identical to CgiError, but also quits with the passed error message.

sub CgiDie {
  local (@msg) = @_;
  &CgiError (@msg);
  die @msg;
}


# PrintVariables
# Nicely formats variables.  Three calling options:
# A non-null associative array - prints the items in that array
# A type-glob - prints the items in the associated assoc array
# nothing - defaults to use %in
# Typical use: &PrintVariables()

sub PrintVariables {
  local (*in) = @_ if @_ == 1;
  local (%in) = @_ if @_ > 1;
  local ($out, $key, $output);
```

```
  $output =  "\n<dl compact>\n";
  foreach $key (sort keys(%in)) {
    foreach (split("\0", $in{$key})) {
      ($out = $_) =~ s/\n/<br>\n/g;
      $output .=  "<dt><b>$key</b>\n <dd>:<i>$out</i>:<br>\n";
    }
  }
  $output .=  "</dl>\n";

  return $output;
}

# PrintEnv
# Nicely formats all environment variables and returns HTML string
sub PrintEnv {
  &PrintVariables(*ENV);
}


# The following lines exist only to avoid warning messages
$cgi_lib'writefiles =  $cgi_lib'writefiles;
$cgi_lib'bufsize    =  $cgi_lib'bufsize ;
$cgi_lib'maxbound   =  $cgi_lib'maxbound;
$cgi_lib'version    =  $cgi_lib'version;
$cgi_lib'filepre    =  $cgi_lib'filepre;

1; #return true
```

Hughes Technologies

Mini SQL 2.0

Beta

Language Specification

Introduction

The mSQL language offers a significant subset of the features provided by ANSI SQL. It allows a program or user to store, manipulate and retrieve data in table structures. It does not support some relational capabilities such as views and nested queries. Although it does not support all the relational operations defined in the ANSI specification, it does provide the capability of "joins" between multiple tables.

The definitions and examples below depict mSQL key words in upper case, but no such restriction is placed on the actual queries.

The Create Clause

The create clause as supported by mSQL 2 can be used to create tables, indices, and sequences. It cannot be used to create other definitions such as views. The three valid constructs of the create clause are shown below:

```
CREATE TABLE table_name (
        col_name col_type [ not null ]
        [ , col_name col_type [ not null ] ]**
)

CREATE [ UNIQUE ] INDEX index_name ON table_name (
        field_name
        [ , field_name ] **
)

CREATE SEQUENCE ON table_name [ STEP step_val ] [ VALUE initial_val ]
```

An example of the creation of a table is show below:

```
CREATE TABLE emp_details (
```

```
    first_name char(15) not null,
    last_name char(15) not null,
    comment text(50),
    dept char(20),
    emp_id int
)
```

The available types are:-

char (len)	String of characters (or other 8 bit data)
text (len)	Variable length string of chracters (or other 8 bit data) The defined length is used to indicate the expected average length of the data. Any data longer than the specified length will be split between the data table and external overflow buffers. **Note** : text fields are slower to access than char fields and cannot be used in an index nor in LIKE tests.
int	Signed integer values
real	Decimal or Scientific Notation real values

The table structure shown in the example would benefit greatly from the creation of some **indices**. It is assumed that the *emp_id* field would be a unique value that is used to identify an employee. Such a field would normally be defined as the primary key. mSQL 2.0 has removed support for the primary key construct within the table creation syntax although the same result can be achieved with an index. Similarly, a common query may be to access an employee based on the combination of the first and last names. A compound index (i.e. constructed from more than 1 field) would improve performance. We could construct these indices using :

```
CREATE UNIQUE INDEX idx1 ON emp_details (emp_id)
CREATE INDEX idx2 ON emp_details (first_name, last_name)
```

These indices will be used automatically whenever a query is sent to the database engine that uses those fields in its WHERE clause. The user is not required to specify any special values in the query to ensure the indices are used to increase performance.

Sequences provide a mechanism via which a sequence value can be maintained by the mSQL server. This allows for atomic operations (such as getting the next sequence value) and removes the concerns associated with performing these operations in client applications. A sequence is associated with a table and a table may contain at most one sequence.

Once a sequence has been created it can be accessed by SELECTing the _seq system variable from the table in which the sequence is defined. For example

```
CREATE SEQUENCE ON TABLE test STEP 1 VALUE 5
SELECT _seq FROM test
```

The above CREATE operation would define a sequence on the table called *test* that had an initial value of 5 and would be incremented each time it is accessed (i.e. have a step of 1). The SELECT statement above would return the value 5. If the SELECT was issued again, a value of 6 would be returned. Each time the _seq field is selected from *test* the current value is returned to the caller

and the sequence value itself is incremented.

Using the STEP and VALUE options a sequence can be created that starts at any specified number and is incremented or decremented by any specified value. The value of a sequence would decrease by 5 each time it was accessed if it was defined with a step of -5.

The Drop Clause

The Drop clause is used to remove a definition from the database. It is most commonly used to remove a table from a database but can also be used for removing several other constructs. In 2.0 it can be used to remove the definition of an index, a sequence, or a table. It should be noted that *dropping* a table or an index removes the data associated with that object as well as the definition.

The syntax of the drop clause as well as examples of its use are given below.

```
DROP TABLE table_name
DROP INDEX index_name FROM table_name
DROP SEQUENCE FROM table_name
```

for example

```
DROP TABLE emp_details
DROP INDEX idx1 FROM emp_details
DROP SEQUENCE FROM emp_details
```

The Insert Clause

Unlike ANSI SQL, you cannot nest a select within an insert (i.e. you cannot insert the data returned by a select). If you do not specify the field names they will be used in the order they were defined - you must specify a value for every field if you do this.

```
INSERT INTO table_name [ ( column [ , column ]** ) ]
    VALUES (value [, value]** )
```

for example

```
INSERT INTO emp_details
    (first_name, last_name, dept, salary)
    VALUES (`David', `Hughes', `Development','12345')

INSERT INTO emp_details
    VALUES (`David', `Hughes', `Development','12345')
```

The number of values supplied must match the number of columns.

The Select Clause

The SELECT offered by mSQL lacks some of the features provided by the standard SQL specification. Development of mSQL 2 is continuing and some of this missing functionality will

be made available in the next beta release. At this point in time, mSQL's select does not provide

- Nested selects
- Implicit functions (e.g. count(), avg())

It does however support:

- Joins - including table aliases
- DISTINCT row selection
- ORDER BY clauses
- Regular expression matching
- Column to Column comparisons in WHERE clauses
- Complex conditions

The formal definition of the syntax for mSQL's select clause is

```
SELECT [table.]column [ , [table.]column ]**
    FROM table [ = alias] [ , table [ = alias] ]**
    [ WHERE [table.] column OPERATOR VALUE
        [ AND | OR [table.]column OPERATOR VALUE]** ]
    [ ORDER BY [table.]column [DESC] [, [table.]column [DESC] ]
```

OPERATOR can be <, >, =, <=, >=, <>, LIKE, RLIKE or CLIKE
VALUE can be a literal value or a column name

Where clauses may contain '(' ')' to nest conditions e.g. "where (age < 20 or age > 30) and sex = 'male'" .

A simple select may be

```
SELECT first_name, last_name FROM emp_details
    WHERE dept = `finance'
```

To sort the returned data in ascending order by last_name and descending order by first_name the query would look like this

```
SELECT first_name, last_name FROM emp_details
    WHERE dept = `finance'
    ORDER BY last_name, first_name DESC
```

And to remove any duplicate rows from the result of the select, the DISTINCT operator could be used:

```
SELECT DISTINCT first_name, last_name FROM emp_details
    WHERE dept = `finance'
    ORDER BY last_name, first_name DESC
```

mSQL provides three regular expression operators for use in *where* comparisons. The standard SQL syntax provides a very simplistic regular expression capability that does not provide the power nor the flexibility UNIX programmers or users will be accustomed to. mSQL supports the "standard" SQL regular expression syntax, via the LIKE operator, but also provide further functionality if it is required. The available regular expression operators are:

- LIKE - the standard SQL regular expression operator.
- CLIKE - a standard LIKE operator that ignores case.
- RLIKE - a complete UNIX regular expression operator.

Note : CLIKE and RLIKE are not standard SQL and may not be available in other

implementations of the language if you decide to port your application. They are however very convenient and powerful features of mSQL.

The regular expression syntax supported by the LIKE and CLIKE operators is that of standard SQL and is outlined below

- `_' matches any single character
- `%' matches 0 or more characters of any value
- `\' escapes special characters (e.g. `\%' matches % and `\\' matches \ )
- all other characters match themselves

As an example of the LIKE operator, it is possible to search for anyone in the finance department who's last name consists of any letter followed by `ughes', such as Hughes. The query to perform this operation could look like

```
SELECT first_name, last_name FROM emp_details
    WHERE dept = `finance' and last_name like `_ughes'
```

The RLIKE operator provides access to the power of the UNIX standard regular expression syntax. The UNIX regular expression syntax provides far greater functionality than SQL's LIKE syntax. The UNIX regex syntax does not use the '_' or '%' characters in the way SQL's regex does (as outlined above). The syntax available in the RLIKE operator is

- '.' matches any single character
- '^' When used as the first charactr in a regex, the caret character forces the match to start at the first character of the string
- '$' When used as the last charactr in a regex, the dollar sign forces the match to end at the last character of the string
- '[ ]' By enclosing a group of single characters withing square brackets, the regex will match a single character from the group of characters. If the ']' character is one of the characters you wish to match you may specifiy it as the first character in the group without closing the group (e.g. '[]abc]' would match any single character that was either ']', 'a', 'b', or 'c'). Ranges of characters can be specified within the group using the 'first-last' syntax (e.g. '[a-z0-9]' would match any lower case letter or a digit). If the first charactr of the group is the '^' character the regex will match any single character that is **not** contained within the group.
- '*' If any regex element is followed by a '*' it will match **zero or more** instances of the regular expression.

The power of a relational query language starts to become apparent when you join tables together during a select operation. Lets say you had two tables defined, one containing staff details and another listing the projects being worked on by each staff member, and each staff member has been assigned an employee number that is unique to that person. You could generate a sorted list of who was working on what project with a query like:

```
SELECT emp_details.first_name, emp_details.last_name, project_details.project
    FROM emp_details, project_details
    WHERE emp_details.emp_id = project_details.emp_id
    ORDER BY emp_details.last_name, emp_details.first_name
```

mSQL places no restriction on the number of tables "joined" during a query so if there were 15 tables all containing information related to an employee ID in some manner, data from each of those tables could be extracted, by a single query. One key point to note regarding joins is that you must qualify all column names with a table name. mSQL does not support the concept of uniquely named columns spanning multiple tables so you are forced to qualify every column

name as soon as you access more than one table in a single select.

mSQL also supports table aliases so that you can perform a join of a table onto itself. This may appear to be an unusual thing to do but it is a very powerful feature if there are rows within a single table relate to each other in some way. An example of such a table could be a list of people including the names of their parents. In such a table there would be multiple rows with a parent/child relationship. Using a table alias you could find out any grandparents contained in the table using something like

```
SELECT t1.parent, t2.child from parent_data=t1, parent_data=t2
    where t1.child = t2.parent
```

The table aliases t1 and t2 both point to the same table (parent_data in this case) and are treated as two different tables that just happen to contain exactly the same data.

The Delete Clause

The SQL DELETE construct is used to remove one or more entries from a database table. The selection of rows to be removed from the table is based on the same *where* construct as used by the SELECT clause. The syntax for mSQL's delete clause is

```
DELETE FROM table_name
    WHERE column OPERATOR value
    [ AND | OR column OPERATOR value ]**
```

OPERATOR can be <, >, =, <=, >=, <>, LIKE, RLIKE, or CLIKE

for example

```
DELETE FROM emp_details WHERE emp_id = 12345
```

The Update Clause

The SQL update clause is used to modify data that is already in the database. The operation is carried out on one or more rows as specified by the *where* construct. The value of any number of fields on the rows matching the where construct can be updated. mSQL places a limitation on the operation of the update clause in that it cannot use a column name as an update value (i.e. you cannot set the value of one field to the current value of another field). Only literal values may by used as an update value. The syntax supported by mSQL is

```
UPDATE table_name SET column=value [ , column=value ]**
    WHERE column OPERATOR value
        [ AND | OR column OPERATOR value ]**
```

OPERATOR can be <, >, =, <=, >=, <>, LIKE, RLIKE or CLIKE

for example

```
UPDATE emp_details SET salary=30000 WHERE emp_id = 1234
```